Strong Gravitational Lens Modelling

Thesis

Strong Gravitational Lens Modelling

Charles Frederick Weiner
School of Physical Sciences, The Open University

Thesis submitted for the degree of MPhil
Supervisors: Prof. Stephen Serjeant & Dr Judith Croston

December 2019

Abstract

The objective of this project is to examine the extent to which strong gravitational lensing can constrain cosmological parameters. I present the results of applying a modified version of an existing model of strong gravitational lensing to forthcoming surveys by Euclid, currently scheduled to launch in 2021, and investigate also how the model may be adapted further to accommodate background galaxy sources that have not previously been included. The initial model, on which the modifications are based, was first constructed by Dr Tom Collett (Collett 2015) with the source code, at the time of writing, freely available on *https://github.com/tcollett/LensPop*. The study commences with a review of the existing model's code, which includes a mapping of the key dependencies. As a natural consequence of this, discrepancies that I have identified within the code are detailed, as are inconsistencies with the supporting article by Collett (2015) in which the principal features of his model are described. Once the discrepancies are corrected, or otherwise resolved, the modified model is run and the implications of these assessed: most are found to be minor, although more significant issues arise when the model is tested under non-standard cosmologies. An analysis of the results for both Euclid's Wide Field and Deep Field surveys is presented using the modified model, as are predictions by the model for the forthcoming Cosmic Evolution Survey (COSMOS) and the Wide Field InfraRed Survey Telescope (WFIRST). In comparison to the Wide Field survey, the model's prediction of a 7-fold increase in the sky density of detectable lenses for the Euclid Deep Field survey is found to be mainly due to an increased sensitivity of 2 magnitudes in the latter. For the COSMOS survey, a prediction of some 120 lenses suggests that further lensing systems have yet to be confirmed in the survey field whereas, in the case of WFIRST, a prediction of just under 100,000 lenses means its increased depth almost compensates for the smaller area, when compared to the Euclid Wide Field survey; compared to the Euclid Deep Field survey, on the other hand, WFIRST is both wider and deeper, with this prediction representing a 25-fold increase in the number of discoverable lenses. The extent to which the model can constrain cosmological parameters is then considered. This requires an investigation not only of the model's direct sensitivity to a cosmology by virtue of

the lensing equations, but also of the model's sensitivity to any astrophysical assumptions, such as those governing density or luminosity functions, that are intrinsic to the code. I find there is prima facie evidence that the model does constrain the cosmologies tested, and conclude also that it is not particularly sensitive to those astrophysical assumptions. Finally, by replacing the simulated source data described in Collett (2015) with a more appropriate mock catalogue, I examine the predictions of the model when submillimetre galaxies are considered. In this respect, a source population comprising solely submillimetre galaxies gives rise to an under-prediction by the model of the number of lenses, when compared to other studies; furthermore, once adapted in this fashion, the model does not impose any significant constraints on the cosmologies tested.

Acknowledgments

I am indebted to all those individuals who have helped and encouraged me at the Open University. In particular, I should like to express my thanks to Professor Stephen Serjeant for allowing me this opportunity for further study, and for his continued support and guidance throughout the project. Stephen's sense of fun, the clarity of his explanations, and his seemingly unlimited patience, have resulted in a learning experience that has been truly enjoyable and fulfilling.

To Harvey, my brother-in-law, my thanks for getting me into this in the first place: without his questions, I may never have gone looking for the answers.

To my darling wife, Adele. I shall always be grateful for her considerable support, her relentless encouragement - and also for her forbearance: it has meant more to me than she probably realises.

And to my Mum and Dad - forever watching over me, and forever an inspiration.

Contents

Chapter 1

Introduction

Abstract

This chapter commences with a discussion of gravitational lensing and its significance as a (testable) consequence of Einstein's General Theory of Relativity. An outline is then presented of some of the main surveys that involve gravitational lensing, both current and forthcoming. Those chosen include the Sloan Lens ACS Survey (SLACS), currently the largest single collection of galaxy-scale strong lenses, the H_0 Lenses in COSMOGRAILS's Wellspring program (H0LiCOW), which applies time-delay methods to measure values for the Hubble constant, and the SpaceWarps program, significant for its use of citizen science in the field. Two forthcoming programs, namely the Cosmic Evolution Survey (COSMOS) and the Wide Field InfraRed Survey Telescope (WFIRST) are described, although these are subject to a more detailed analysis in a later chapter, as well as the Large Synoptic Survey Telescope (LSST), a ground-based observatory scheduled to commence operations in 2022 and designed to produce an unprecedented volume of astronomical data for both scientists and the public. After a comment on non-optical surveys, the chapter concludes with a description of the Euclid mission, which lies at the core of this project and is scheduled for launch in 2021; the Euclid surveys are expected to detect several orders of magnitude more galaxy-scale lenses and related phenomena than the total of all previous surveys.

1.1 Historical Context

Gravitational lensing is a consequence of one of the most famous predictions of Einstein's General Relativity - the notion that light is bent in a gravitational field.

In 1919, Eddington and his expedition were able to confirm this observationally, by measuring the deflection of starlight grazing the Solar limb during a Solar eclipse; see Figure 1.1. The publicity that followed made Albert Einstein a household name. Several earlier attempts (most notably by Erwin Finlay Freundlich of the Berlin Observatory, and by William Campbell of the Lick Observatory) had been made between 1911 and 1915, but these had been thwarted by external issues, not least of which were poor weather conditions and the outbreak of World War One.

As it happens, it was fortuitous the earlier attempts could not be completed, because Einstein at the time had miscalculated the effect and his prediction would have been wrong. The nature of the initial miscalculation is worth noting. Einstein's calculation was equivalent to determining the deflection of starlight just by applying Newtonian theory to a particle moving at the speed of light (Will 2015). The correction was needed because a (further) bending of light due to *spatial curvature* must be taken into account - a feature of General Relativity which Einstein himself had effectively overlooked. The two effects have the same magnitude, which means Einstein's 1919 prediction would have been out by a factor of 2 had he not corrected it. It is also worth pointing out that the deflection of light by gravity was actually postulated much earlier than this: the first written account was by Soldner (1804), whose calculations (understandably) relied on Newtonian gravity alone and were therefore similarly incorrect.

In the event, Einstein corrected his calculations in time, and Eddington was able to confirm the revised prediction of a deflection of 1.74 arcsecs to within 20%, bringing Einstein's 'weird theory of non-Euclidean space' to the attention of scientists and the general public alike (Eddington 1919).

It is this deflection of light by massive bodies, and the resultant phenomena, that is now referred to as *gravitational lensing*.

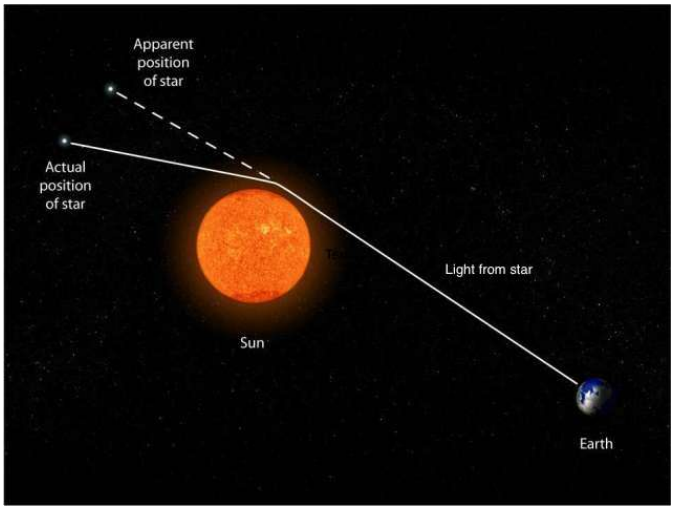

Figure 1.1: Schematic Deflection of Starlight
(adapted from https://electrolights.wordpress.com)

As one of the classical tests of General Relativity, light bending is of particular importance because it was predicted by the theory before it was observed. In fact, the first detection on a cosmological scale did not occur until 1979, when Walsh et al. (1979) discovered the 'double quasar' Q0957+561: two quasar images separated by 5.7 arcsecs, at a similar redshift (z=1.405), and with very similar spectra. Quasars are amongst the rarest objects in the universe, so the probability of locating two of them so close together is extremely low. The similarity of the spectra of the two images, as well as the presence of a foreground galaxy between them, led to the conclusion that this was actually just *one* quasar, but that intervening matter was responsible for bending the light to produce the two separate images.

Whereas most astronomy can investigate only luminous matter, gravitational lensing is a purely gravitational effect; the phenomenon is created only by the intervening matter distribution, regardless of whether it is luminous or not. Since the discovery of the first gravitational lens, gravitational lensing has consequently come to be recognised as a major tool for mapping the distribution of mass in the universe, and for searching for dark matter, dark energy, and compact objects. It has been used to study the physics of quasars and the internal structure of galaxies,

and for the detection of extra-solar planets (Ellis et al. 2012). More recently, gravitational lensing has also become an important factor in the study of gravitational waves emitted by sources at cosmological distances (Li, Shun-Sheng. et al. 2018).

Finally, gravitational lensing has given rise to some of the most beautiful images in cosmology. The many observations to date include multiple images of a single source, luminous arcs, and 'Einstein rings' (where the image fills an anulus around the lens); see Figure 1.2. We see in all of these the direct effect that matter can have on the curvature of spacetime around it: evidence not only in support of General Relativity, but testament also to gravitational lensing "as Einstein's gift to astronomy" (Will 2015).

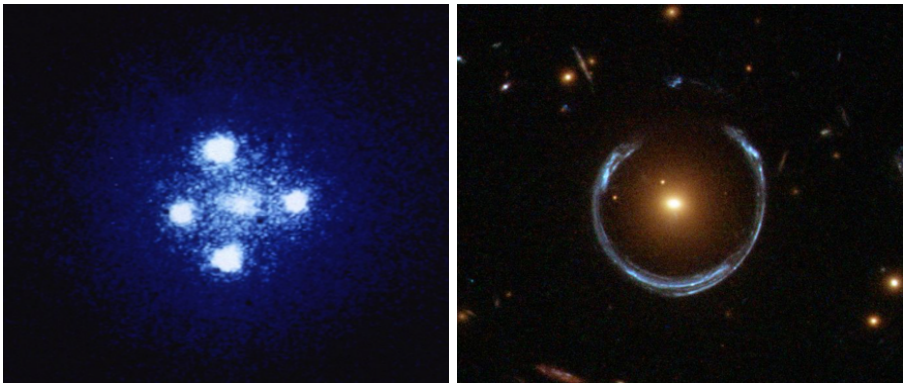

Figure 1.2: Examples of a lensed quasar (left) and an Einstein Ring (right) *(images from HST)*

1.2 Recent & Forthcoming Surveys

This project is primarily concerned with the forthcoming Euclid survey mission (Laureijs et al. 2011). Whilst a description of that mission is presented in the next section, it is helpful to outline here examples of other recent and forthcoming surveys, in order to highlight the role of gravitational lensing in modern cosmology. The examples chosen include early surveys such as SLACS (Bolton et al. 2006), currently the largest single collection of galaxy-scale strong lenses, and H0LiCOW (Suyu et al. 2017), which applies time-delay methods to lensed systems as a

means of measuring the Hubble constant. Also discussed is the SpaceWarps program (Marshall et al. 2015), which has pioneered the use of citizen science in the search for, and analysis of, gravitational lenses. Two forthcoming surveys, namely COSMOS (Scoville et al. 2007) and WFIRST (Green et al. 2012), are described although these are the subject of a later chapter, when they will be further discussed in the context of an application of the model by Collett (2015). After an outline of LSST (Abell et al. 2009, Ivezic et al. 2008), which is a ground-based observatory scheduled to commence operations in 2022 and expected to make available an unprecedented volume of astronomical data to both scientists and members of the public worldwide, this section will conclude with a comment on non-optical surveys.

SLACS

The Sloan Lens ACS (SLACS) Survey was set up as a project to combine the massive data volume of the Sloan Digital Sky Survey (SDSS) with the high-resolution imaging capability of the Hubble Space Telescope, to both identify and study a large and uniform sample of galaxy-scale strong gravitational lenses.

The survey uses the spectroscopic database of SDSS to identify lens candidates. Specifically, the strategy is firstly to examine the SDSS galaxy spectra to identify emission lines not associated with the primary target galaxy, but with an additional source aligned with the first galaxy and located at a higher redshift. Bolton et al. (2006) find spectra such as these occur with a frequency of between 1 in 500 and 1 in 1,000; with nearly a million galaxy spectra, a database such as that of SDSS is key to obtaining a statistically significant sample.

The lens candidates are then ranked in terms of their probability of being lensing systems and observed with the HST/ACS, revealing in some cases the image of the more distant galaxy distorted into an Einstein ring. By measuring the angular size of Einstein rings, in combination with distances measured from the SDSS spectra, the total masses interior to the rings can be determined from lensing geometry. Combining these with measurements of the sizes, brightness,

and stellar velocities of those galaxies finally enables an analysis of their structure and evolution.

According to the most recent data release (Auger et al. 2009), SLACS has identified nearly 100 lenses and lens candidates. Approximately 80% of the grade 'A' (genuine) lensing systems have lenses with elliptical morphologies while ∼10% show spiral structure; the remaining lenses have lenticular morphologies. It also appears that SLACS lenses have total mass distributions that are uniform across a wide range in cosmic time and lens mass and, significantly, are therefore well approximated by isothermal ellipsoids (Treu et al. 2006, Ellis 2010).

With spectroscopic redshifts for both lens and source available for every system, SLACS is the largest homogeneous dataset of galaxy-scale strong lensing systems assembled to date. According to Auger et al. (2009), SLACS lenses are representative of the overall population of massive early-type galaxies with $M_* \geq 10^{11} M_\odot$, and an ideal dataset to investigate the kpc-scale distribution of luminous and dark matter in galaxies out to z $\sim$ 0.5.

H0LiCOW

The H_0 Lenses in COSMOGRAIL's Wellspring (H0LiCOW) program is part of the Cosmological Monitoring of Gravitational Lensing (COSMOGRAIL) collaboration (Eigenbrod et al. 2005), which has been monitoring about 20 lensed quasars since 2004 with a combination of ground-based and space-based telescopes. These include the Hubble Space Telescope, the Spitzer Space Telescope, the Subaru Telescope, the Canada-France-Hawaii Telescope, the Gemini Observatory, and the W. M. Keck Observatory.

The program was created with the aim of determining a value for the Hubble constant H_0 by measuring the time delays between multiple images of a strongly lensed quasar. The method was first proposed by Refsdal (1964), before strong gravitational lenses had even been discovered. The theory behind this approach is that since light from a quasar fluctuates over time, then so too will the light from its lensed images. The crucial point is that the fluctuations in the lensed

images will be observed at different times, corresponding to the differences in the paths travelled by their respective light rays.

Gravitational lensing theory is the subject of the next chapter, but at this stage it is worth noting that the time delay Δt depends on both the 'time-delay distance' $D_{\Delta t}$ and the distribution of the lens mass such that

$$\Delta t = D_{\Delta t} \frac{\Delta \Phi}{c}$$

where $\Delta \Phi$ is the Fermat potential difference determined by the lens mass distribution.

By measuring the time delay from photometric light curves of the quasar images, and by modelling the mass distribution of the lens, the time-delay distance to the lensing system can be calculated and applied together with the distance-redshift relation to study the underlying cosmology, which in turn is sensitive to values of the Hubble constant.

Looking ahead to the chapter on gravitational lensing theory, we may note that a more precise definition for the time delay distance is given by

$$D_{\Delta t} \equiv (1 + zl) \frac{D_L D_S}{D_{LS}}$$

where D_L, D_{LS} and D_S denote the angular diameter distances between the observer and lens, the lens and source, and the observer and source, respectively and zl is the lens redshift. The time delay distance is primarily sensitive to H_0 as a result of its dependence on the three angular diameter distances (Suyu et al. 2013).

In their most recent results, the H0LiCOW collaboration analysed four quasar systems that had been multiple-imaged through strong gravitational lensing and derived a value of $H_0 = 72.5^{+2.1}_{-2.3}$ km/s/Mpc, which is a precision of about 3%. This is completely independent of, and consistent with, other measurements such as those of the SH0ES project[1] (e.g. Riess et al. 2011, 2016) where Cepheid variable stars and supernovae were used as the points of reference. Significantly, however,

[1] '*Supernovae and H0 for the Equation of State of dark energy*'.

combining these into a single measurement gives a value for H_0 that is 4.2σ higher than the most recent CMB prediction from the Planck satellite and 3.4σ than the galaxy clustering and weak lensing measurement from the DES collaboration (*http://shsuyu.github.io/H0LiCOW/site/*). Given the assumption of a standard flat ΛCDM cosmology in deriving those measurements, this anomaly points to a potentially fundamental flaw in that cosmological model, and highlights the importance of continuing to study cosmological distances from the many time-delay lenses that H0LiCOW anticipate from forthcoming surveys.

SpaceWarps

The SpaceWarps program represents the introduction of citizen science into the search for gravitational lenses. Such a search is often compared to looking for a needle in a haystack, and several automated lens-finding algorithms have been - or are in the course of being - developed to take on this task, e.g. Brault & Gavazzi (2015). There are many problems facing these however, not least of which are the similarity of many lensed images to features commonly found in galaxies and to simple imaging artefacts. As a result, many automated lens-finding algorithms are prone to a high rate of false positive detections.

Whilst the complexity of gravitational lenses renders their detection particularly difficult for automated processes, this is not necessarily the case for detection by the human eye. However, although humans are generally capable of dealing with such complexities, a manual approach to visually combing the 'haystack' for candidates is neither ideal nor practical for individuals acting alone. Following on from the success of the Galaxy Zoo project (Lintott et al. 2008), the SpaceWarps program was established to call on citizen science instead to search through the data, since this is a task that naturally lends itself to visual identification by a large community of volunteers.

The first lens search by SpaceWarps used data from the Canada-France-Hawaii-Telescope Legacy Survey (CFHTLS) (Heymans et al. 2012). In the first stage of the program, the 160 sq.deg. of imaging was divided into some 430,000 overlapping 82 x 82 arcsec tiles. These were displayed

on the SpaceWarps website, where an inspection by around 37,000 volunteers contributed 11 million image classifications over the course of 8 months, identifying ∼3,000 images as potential candidates. In the second stage, which involved a careful inspection of those candidates, the sample was refined to ∼500, suitable for further inspection by a team of experts (Marshall et al. 2015).

Based on these findings, the SpaceWarps team was able to report the discovery of 29 promising (and 59 total) new gravitational lens candidates for CFHTLS through citizen science (More et al. 2015). Whilst it is true that the citizen scientists found lens candidates that previous algorithms had failed to detect, they recovered only 65% of known lenses. Better training and performance calibration, however, could be a significant factor in improving this figure to 80%. In any case, with classifications by volunteers at rates of between 10^3 and 10^4 images per hour, visual inspection of tens of thousands of images could be performed in just a few weeks, suggesting that citizen science will prove a valuable tool in future searches.

At the time of writing, SpaceWarps is conducting a citizen science search using data from the Hyper Suprime-Cam (HSC). This is a large mosaic CCD camera, attached at the prime focus of the Subaru Telescope on Mauna Kea. In September, results from HSC's first year of data produced the deepest wide field map of the three-dimensional distribution of matter in the universe ever made, allowing researchers to measure the gravitational distortion in images of about 10 million galaxies (*www.sciencedaily.com/releases/2018/09/180926082711.htm*). So far, only a fraction of the available data from HSC has been searched for lenses, using automatic algorithms. Based on the experience of the CFHTLS data, this is an excellent example of an opportunity to benefit again from an application of citizen science.

COSMOS

The Cosmic Evolution Survey (COSMOS) is a deep, wide area, multi-wavelength survey measuring the evolution of galaxies on scales from a few kpc to tens of Mpc. The primary goal is to study the relationship between large scale structure in the universe and dark matter, the

formation of galaxies, and nuclear activity in galaxies (see, for example, Scoville et al. 2007). The main data set covers a 2 square degree equatorial field, and corresponds to a patch of sky about 16 times the size of the full moon in the constellation Sextants (a region of sky chosen as there are few stars and no clouds of gas in our galaxy to obstruct the view). The field has been observed at all accessible wavelengths from X-ray to radio with most of the major space-based and ground-based telescopes. Over 2 million galaxies have been detected, spanning 75% of the age of the Universe.

The first telescope used for COSMOS was the Hubble Space Telescope (HST); this was the largest patch of sky that had ever been covered by HST.

The remit of the survey includes the study of distortions in the shapes of background galaxies that arise from weak gravitational lensing by foreground structures. The reliability of these findings depends, amongst other things, on the instrumental point spread functions. In this respect, the HST-ACS allows for an extraction of shapes for $\sim$87 galaxies per arcmin, which is 2-3 times more than those obtained from the best ground-based data.

The relevance of COSMOS to the identification of strong gravitational lenses will be discussed as a separate topic in a later section of this project, when it will be considered in the context of both an application of Collett's model and in the light of existing studies such as those by Faure et al. (2008) and Jackson (2008).

WFIRST

The Wide Field InfraRed Survey Telescope (WFIRST) is an observatory designed by NASA for both dark energy research and exoplanet detection. The 6 year mission is scheduled to launch in the mid-2020s. The telescope itself has a 2.4m primary mirror, the same size as the HST primary mirror, and acts as the 'front end' to two instruments: the Wide Field Instrument and the Coronagraph Instrument.

The 300-megapixel Wide Field Instrument will have a field of view 100 times greater than the Hubble infra-red instrument; an implication of this, for example, is that whilst the HST/WFC3/IR PHAT survey required 432 pointings to cover M31, only 2 pointings will be required by WFIRST. In addition to measuring the matter in hundreds of millions of distant galaxies through a form of lensing known as *weak* gravitational lensing, WFIRST will use another form of lensing known as *micro*lensing to search the inner Milky Way for exoplanets. (The different forms of gravitational lensing will be explained in the next chapter). The microlensing survey will monitor 100 million stars for hundreds of days, and is expected to find about 2,500 exoplanets; this method is sensitive enough to find planets smaller than Mars, orbiting their host stars at distances ranging from closer than Venus to beyond Pluto.

The Coronagraph Instrument will perform high contrast imaging and integral field spectroscopy, which will permit the identification of dim planets in the vicinity of bright stars. As NASA's first advanced coronagraph in space, it will be 1,000 times more powerful than any flown previously and will image gas giant planets orbiting mature Sun-like stars, allowing them to be analysed to a degree not previously possible.

In a later section, we will consider an application of Collett's model to predict the (strong) lensing systems discoverable by WFIRST. As far as data from the actual mission is concerned however, a single WFIRST image will contain over a million galaxies; in this regard, the mission stands as another example of how an unprecedented wealth of search results, obtained largely through the phenomena of gravitational lensing, could be combined with citizen science to address key cosmological questions. (Further details of this mission may be found in Green et al. 2012).

LSST[2]

The Large Synoptic Survey Telescope (LSST) is a large, wide-field ground-based observatory designed to obtain repeated images covering the sky visible from Cerro Pachón in northern Chile.

[2] *https://www.lsst.org*

It is scheduled to commence operations in 2022, with a stated mission to build a 'well-understood system that provides a vast astronomical dataset for unprecedented discovery of the deep and dynamic universe': it will do this by conducting a 10-year survey of the sky, to deliver a relational database of about 32 trillion observations of 40 billion objects and an astronomical catalogue thousands of times larger than any previous program. Data from LSST is intended ultimately to be available to scientists and to the public around the world.

The telescope will have an 8.4 m (6.5 m effective) primary mirror, and a 3.2 gigapixel camera. The special three-mirror design will afford an exceptionally wide field of view of 9.6 sq.deg. The standard observing sequence will consist of pairs of 15-second exposures in a given field, making LSST capable of imaging about 10,000 sq.deg. of sky in a single filter in three clear nights; a single exposure will cover 49 times the area of the Moon. The survey will yield contiguous overlapping imaging of half the sky in six optical bands, with each sky location visited close to 1,000 times over its lifetime. By continually surveying the sky and identifying changes in real time, LSST will effectively provide the first ever high-definition colour movie of the deep universe.

Underpinning the LSST program are four science objectives: (a) probing dark energy and dark matter, (b) taking an inventory of the Solar System, (c) exploring the transient optical sky, and (d) mapping the Milky Way.

The role of gravitational lensing is particularly significant with regard to the first of those objectives, namely, probing dark energy and dark matter. Dark energy manifests itself in two ways. The first is the relationship between redshift and distance, and the second is the rate at which matter clusters with time: the accelerated expansion of the Universe, caused by dark energy, opposes the gravitational attraction that would otherwise lead to increased clumping of dark matter structures.

Gravitational lensing by clusters enables the mass distribution in clusters to be explored. Many lensed arcs have been discovered in clusters, and the number of lensed arcs in a cluster is a function of the cluster mass; the majority of massive clusters ($> 10^{15} M_{\odot}$) exhibit strongly lensed

background galaxies when observed up to depths that will be achievable by LSST. Combined with other (weak) lensing measurements, the density profile of clusters over a wide range in radii may be used to map the distribution of mass as a function of redshift, tracing the history of both the expansion of the universe and the growth of structure, which in turn can be used to constrain the dynamical behaviour of dark energy. Being able to identify systems of multiple images via their colours and morphologies requires high resolution imaging: this is one of the most significant technical constraints on LSST image quality, and one in which LSST will have an advantage over precursor surveys like the Dark Energy Survey (DES) (Abbott et al. 2016). The number of multiple image systems detectable with LSST is likely to be $\sim 1,000$, although the more massive clusters are likely to display many multiple image systems. Given the relative scarcity of those massive clusters however, LSST is likely to identify a sample of around 1,000 (strong) lensing clusters, with the majority displaying a single multiple image system.

As far as galaxy-scale (strong) lensing is concerned, LSST is expected to find $\sim 2,600$ lensed quasars. These will mainly be at $z \sim$ 2-3 with lensing galaxies typically at $z \sim 0.6$ (although a significant fraction of lensing is likely to be produced by galaxies at $z > 1$). This will be nearly two orders of magnitude larger than the current largest survey of lensed quasars (Treu et al. 2018). LSST is also expected to identify 330 lensed supernovae, with lenses primarily in the form of massive elliptical galaxies at $z \sim 0.2$; with light profiles of supernovae well understood, these observations will enable accurate measurements to be made of time delays between successive LSST images.

A unique aspect of LSST for exploring dark energy and dark matter is the application of multiple cross-checking probes to improve the level of precision. Any one probe on its own will constrain combinations of cosmological parameters, but these will be degenerate. Additionally, each probe is affected by different systematics. A combination of probes, on the other hand, will permit systematics to be calibrated and degeneracies to be broken, resulting in a more robust set of constraints. An example of this is the joint analysis by LSST of (weak) lensing and baryonic acoustic oscillations (BAO), which will significantly tighten existing constraints on the dark energy equation of state. By way of brief explanation, BAOs are a feature arising from the interaction of

baryons with photons prior to the epoch of decoupling (about 400,000 years after the Big Bang). This interaction produced a pressure force that opposed the gravitational attraction of baryons in areas of high dark matter density. The restorative nature of the radiation pressure against the gravitational attraction caused the baryons to oscillate - compressing and rarefacting - in the gravitational potential well of the dark matter, in a manner analogous to sound or acoustic waves. These oscillations continued until decoupling occurred, at which point the photons no longer interacted with baryons and the pressure support was removed. The subsequent gravitational collapse resulted ultimately in the formation of galaxy structures, reflected in the galaxy power spectrum: galaxy structure was enhanced in regions where the oscillations were at maximum compression at the time the pressure support was removed, whereas it was suppressed for those at rarefaction (e.g. Eisenstein 2005). BAO and (weak) lensing techniques each have their own systematics and parameter degeneracies. When shear power spectra (lensing) and galaxy power spectra (BAO) techniques are analysed jointly, extra information is obtained from the cross power spectra which cannot be captured from either technique in isolation. Additionally, the two methods can mutually calibrate some of their systematics: for example, BAO can be applied to self-calibrate the photometric redshift error distribution, which is one of the most critical systematics for lensing tomography since the true-redshift distribution of galaxies in each photometric redshift bin must be known accurately to interpret lensing data correctly.

Gravitational lensing also has a role to play in the exploration of the transient optical sky. In this context, it is the ability to detect lensing *events* that is relevant: the observable is a *temporal* variability due to the relative motion of source, lens, and observer. LSST is expected to be a major contributor in the detection of this form of (micro)lensing by virtue of, amongst other things, its large area coverage, which will enhance the probability of detecting rare events: it will probe the Galactic halo, placing constraints on the existence of massive compact halo objects (MACHOs) - building on the collaborations of the late 1990s (e.g. Alcock et al. 1997) - and exploring stellar populations of the halo, and also detect lensing of stars in a wide range of external galaxies in addition to our own. It is worth noting also that any deviation of a lensing event from point-source and point-lens geometry tends to be long-lasting. By sampling the light curve with good photometric sensitivity at several points, LSST will therefore be able to

identify its unique features to analyse the physical characteristics of the lens. Finally, multiple images whose positions and intensities change as the event progresses can result in astrometric shifts. For nearby lenses, these shifts can be several milli-arcsecs. Since these are expected to be measurable by LSST (Abell et al. 2009, p.273), a combination of astrometric and photometric monitoring represent another aspect in which the program is likely to prove of value.

(A comprehensive description of both the design and objectives of the LSST program may be found in Abell et al. 2009, Ivezic et al. 2008).

Non-optical surveys

The surveys detailed so far in this chapter are predominantly optical surveys, mentioned either to help put into context the role played by gravitational lensing, or because of their direct relevance to this project.

It is important to recognise however that there have been a number of non-optical surveys, such as those in the radio or submillimetre frequencies. Examples of these include the Cosmic Lens All-Sky Survey (CLASS), the Herschel Astrophysical Tetrahertz Large Area Survey (H-ATLAS), and the Herschel Multi-tiered Extragalactic Survey (HerMES).

The CLASS survey was established to search for gravitationally lensed compact radio sources (*http://www.jb.man.ac.uk/research/gravlens/class/*). The procedure called for the initial identification of candidates (about 11,000) with the Very Large Array (VLA), based on their flat radio spectra and flux density, then re-observing (and filtering) the candidates at higher resolution with the Multi-Element Radio Linked Interferometer Network (MERLIN), before finally re-observing (and filtering) the surviving candidates at even higher resolution with the Very Long Baseline Array (VLBA). CLASS has so far identified 22 gravitational lens systems, with some of them subsequently monitored to determine the time-delay between images in order to measure the Hubble constant. The last of the CLASS lenses (B0631+519) exhibits complex radio structure

over scales from 3.6 mas to 1.16 arcsec and possesses a nearly complete infra-red Einstein ring; its particularly rich lensed image structures make it ideal for probing the mass properties of the lensing galaxy (York et al. 2005). It should also be noted that CLASS was designed to be a survey with well-defined statistical properties, intended to assist in determining the values of cosmological parameters.

The second of the surveys listed above, H-ATLAS, was an astronomical project awarded on ESA's Herschel Space Observatory and designed to survey 550 square degrees (four times larger than all the other Herschel extragalactic surveys combined). With its PACS[3] and SPIRE[4] cameras, the observatory was not only the largest and most powerful infra-red telescope ever flown in space, but also the first space observatory to cover the entire range from far infra-red to submillimetre wavelengths. In a later chapter of this project, reference will be made to a catalogue of 80 candidate strongly lensed galaxies identified by the H-ATLAS team in the sub-millimetre range (Negrello et al. 2017), where it may be noted that the boost in luminosity and gain in spatial resolution from lensing has enabled a study of the dynamical and morphological properties of star forming galaxies at z $\sim$ 2 to an unprecedented level of detail (*https://www.h-atlas.org/results/highlights/gravitationally-lensed-galaxies-h-atlas*).

Finally, whilst H-ATLAS was awarded in open time, HerMES was a survey awarded on ESA's Herschel Space Observatory in guaranteed time. At 900 hours, HerMES was the largest project on the observatory (*http://hedam.lam.fr*) and was designed for a multi-wavelength understanding of galaxy formation and evolution (Oliver et al. 2012). The project team comprised mainly individuals who had contributed to the development of the SPIRE instrument, used to carry out the observations with a photometer operating at 250μm, 350μm and 500μm. Importantly, HerMES has also been a source of strong gravitationally lensed systems (e.g. Wardlow et al. 2012).

[3] *Photodetector Array Camera and Spectrometer*
[4] *Spectral and Photometric Imaging Receiver*

1.3 The Euclid Surveys

Euclid is a survey mission of the European Space Agency, now scheduled for launch in 2021. It has been designed to investigate the expansion of the Universe, principally by mapping the geometry of the dark universe and the cosmic history of large-scale structure formation.

A comprehensive and detailed description is contained in the Euclid Definition Study Report (Laureijs et al. 2011). By way of overview and context, some of the main features of that study are outlined in this section, with the key questions that Euclid is intended to address about the dark universe summarised in table 1.1.

Table 1.1: Euclid Primary Goals - Key Questions
source: Laureijs et al. (2011)

Topic	Questions
Dynamical Dark Energy	Is dark energy a cosmological constant? Or is it a field that evolves dynamically with the expansion of the Universe?
Modification of Gravity	Is the apparent acceleration a manifestation of a breakdown in general relativity on the largest scales, or a failure of the cosmological assumptions of homogeneity and isotropy?
Dark Matter	What is dark matter? What is the absolute neutrino mass scale and what is the number of relativistic species in the Universe?
Initial Conditions	What is the power spectrum of primordial density fluctuations, which seeded large-scale structures, and are they described by a Gaussian probability distribution?

In its Wide Field mission, Euclid will survey 15,000 sq.deg. of the extra-galactic sky to a depth of 24.5 mag. High-quality data will be provided by means of an imager at visible wavelengths (VIS) and a spectro-photometer in the near infra-red (NISP). Euclid will directly map the dark matter distribution in the Universe through weak gravitational lensing by imaging 1.5 billion galaxies with HST-like resolution. At the same time, it will carry out a spectroscopic redshift

survey of 50 million galaxies over a volume 500 times larger than the SDSS, observing galaxies over 75% of the lifetime of the Universe. In 7 years of mission, the survey is expected to cover 100 times more sky than the HST has done since its launch; see Figure 1.3.

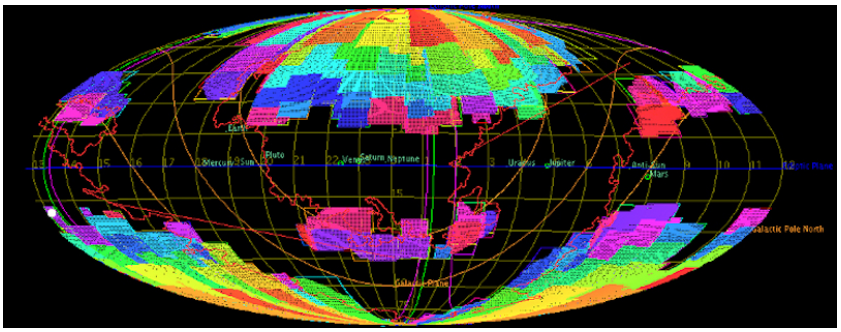

Figure 1.3: Euclid Coverage
The different colours are associated with a six-month period in which the whole sky is accessible by Euclid. During the primary mission there are 12 such periods, indicated here by 12 different colours.
(copyright ESA/Euclid Consortium)

In addition to the Wide Field survey, Euclid will perform a deeper survey over 40 sq.deg. At up to 26.5 mag, the Deep Field survey will be 2 magnitudes deeper than the Wide Field, resulting in a unique survey that is some 50 times larger than the NIR UltraVista survey (McCracken et al. 2013), and 3 times larger and 2 magnitudes fainter than the NIR VIDEO survey (Jarvis et al. 2012). The Deep Field survey will allow the detection of high-redshift star forming galaxies at redshift $z > 7$; it will measure the faint end slope of the H_α luminosity function at all redshifts for which it is detectable, and enable galaxies to be related to their dark matter halos for normal galaxies at redshift $z \sim 2$.

The *Concordance* cosmological model is widely accepted by cosmologists as the standard model for describing the universe (e.g. Ellis et al. 2012). A particular problem with it however is the existence of two dominant components which - to all intents and purposes - remain a complete mystery: namely, dark energy, thought to comprise about 76% of the overall mass-energy density of the Universe and responsible for its accelerated expansion, and dark matter, thought to make up another 20%, and which has a gravitational interaction like normal matter but does not inter-

act with light. Whilst there are many theories as to the nature of both these components, dark energy arguably poses the most fundamental puzzle, with implications for modern theoretical physics that are likely to be profound. In particular, if dark energy behaves as predicted by Einstein's cosmological constant, then the value of the constant is at least 10^{60} times smaller than that predicted from theory (see, for example, the account by Bousso 2012) - the largest discrepancy between theory and observation ever encountered in modern physics - which means that either the cosmological constant is not a correct description of dark energy, or fundamental theories such as quantum field theory and General Relativity need to be overhauled.

By measuring the expansion history and growth of large-scale structure with sufficient precision, it is hoped Euclid will enable us to distinguish time-evolving dark energy models from a cosmological constant and test the theory of gravity on cosmological scales. These same measurements will also allow constraints to be imposed on the initial conditions for the very early Universe, thereby offering some insight into how the Universe began. Indeed, the scientific impact of Euclid is not limited to cosmology, and its unique combination of high-resolution optical imaging, multi-band NIR imaging and spectroscopy up to $z \sim 2$ over most of the extra-galactic sky, is expected to contribute to a vast range of non-cosmology science: the Euclid Consortium comprises some 1,500 registered members, of which more than 900 are researchers in cosmology, astrophysics, theoretical physics, and particle physics.

Finally, and of direct relevance to this project, mention should be made of strong gravitational lensing. Strong gravitational lensing provides for precise measurements of the mass of individual lenses and has a broad range of cosmological and astrophysical applications. However, strong gravitational lensing systems are extremely rare events. Although the number of known galaxies acting as strong gravitational lenses has risen from a handful to hundreds in the past decade or so, finding one typically requires inspection of potentially thousands of pre-selected targets. With its combination of large area and high quality optical images, Euclid is ideally placed to survey for strong gravitational lensing systems. Compared to the total of all previous surveys, it is anticipated that several orders of magnitude more galaxy-scale lenses, arcs and multiply-imaged quasars, will be detected by Euclid.

Strong gravitational lensing and the Euclid surveys lie at the core of this project, and will be discussed in further detail in forthcoming sections. In particular, a key objective is the application to the Euclid survey of the model described in Collett (2015) in order to ascertain the extent to which strong gravitational lensing can constrain cosmological parameters.

Chapter 2

Theory of Gravitational Lensing

Abstract

The chapter commences with a description of the three main forms of gravitational lensing, which include the *strong* form of lensing that is the subject of this project. I then proceed to discuss the theory behind the phenomenon. There are numerous texts and articles covering the topic in detail (e.g. Narayan & Bartelmann 1996, Serjeant 2010), and it is therefore not my intention to present here a comprehensive review of the theory. Instead, in this chapter I focus only on the theoretical features and concepts that are relevant to the core of this project, such as the lensing equation and the assumptions that lie behind it. The chapter concludes with a summary of the key applications of gravitational lensing to modern cosmology.

2.1 Forms of Gravitational Lensing

In a nutshell, gravitational lensing is said to take place when a massive object lies in between a background source and an observer, causing a distortion or magnification (or both) of the image of the source; see Figure 2.1.

By convention, gravitational lensing takes three main forms:-

- *Strong* gravitational lensing occurs when a massive foreground object (typically a galaxy) is sufficiently aligned with an observer and a background source such that the deflection

is comparatively large and the corresponding lens equation has multiple solutions. As a consequence, there will be visible distortions such as multiple images, an Einstein ring, or arcs. This form of gravitational lensing is the primary subject of the model by Collett (2015) and of this project.

- *Weak* gravitational lensing occurs when the lens is a large mass, but the alignment is such that the image of the source is only mildly distorted, displaying a shear effect: the image of the source is smeared into an arc centred on the centre of the lens. The non-random alignment of background sources (there is a tendency for their ellipticities to align) means that shear can be measured statistically, and lensing thus identified, even when distortions of individual sources are too small to be identified.

- *Microlensing* takes place when the lens is a small mass (typically a star) and the distortion of the source image cannot be resolved, regardless of how favourable the alignment. However, the source, lens and observer all have proper relative motions, so any alignment is temporary: a microlensing event can therefore be recognised by the temporary brightening of the combined signal from the source and lens, as the latter passes in front of the former. The timescale of this brightening can range from seconds to years, with information as to the lens mass, and the relative distances and motion, provided by observations of the light curve (although many stars may intrinsically be variable in their output, which can complicate such searches).

2.2 The Geometry of Gravitational Lensing

The geometry of a typical gravitational lens system is illustrated in Figure 2.2. In the illustration, light from a source S is deflected by an angle $\hat{\alpha}$ at the lens L, before it reaches an observer at O. The apparent positions of the source as seen by the observer are at S_1 and S_2 (although for the sake of clarity, the light rays at S_2 are not shown). We note the angle between the optic axis (defined as the line perpendicular to the lens and source planes and passing through the observer) and the true source position is given by β, whereas the angle between the optic axis and the apparent position S_1 is given by θ. The source is located at a transverse distance η from

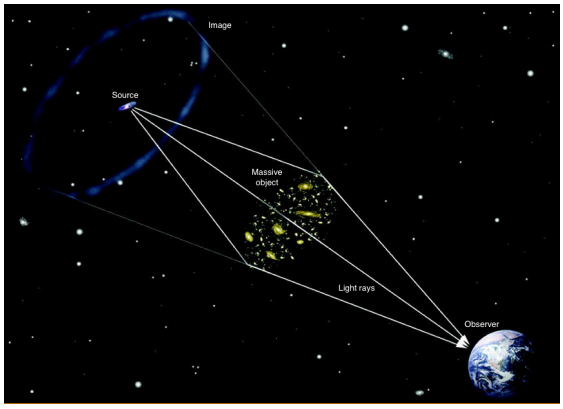

Figure 2.1: Schematic Example of Gravitational Lensing
(adapted from www.theskepticsguide.org)

the optic axis, and the symbol ξ represents the impact parameter of the light as it passes by the lens. The *angular diameter* distances between the observer and lens, the lens and source, and the observer and source, are given by D_L, D_{LS} and D_S respectively.

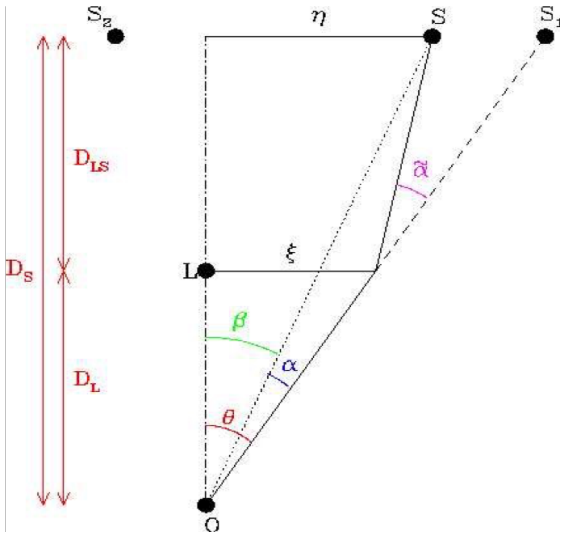

Figure 2.2: The Geometry of a Gravitational Lens System
(adapted from Serjeant 2010)

It is helpful to make two approximations at this stage. Firstly, if the lens is much smaller than the distances to the observer and to the source, then very little time is spent by the photons

undergoing deflection compared to their total travel time. This allows us to use the *thin lens approximation* and assume any change in direction is instantaneous. Secondly, we assume the angles are aways very small, so that we can freely use approximations such as $sin\alpha = \alpha$ or, in the case of our illustration, $\xi = D_L \tan\theta \simeq D_L\theta$. We also note, by way of an important caveat, that the distances in the figure are *not* additive. This is because angular diameter distances are dependant on spacetime curvature, so the Euclidean relationship $D_S = D_{LS} + D_L$ cannot be guaranteed; this dependence will be addressed again in the chapter on cosmological constraints.

Figure 2.2 represents an axially symmetric lens: all light rays from the source to the observer lie in the plane spanned by the centre of the lens, the source and the observer. The deflection angle may therefore be described in one dimension, allowing us to write:

$$\beta = \theta - \alpha \tag{2.1}$$

In general, however, a lens may not be symmetrical and the deflection angle will be a two dimensional vector. In such cases, the angles must be treated as vectors and this becomes:

$$\vec{\beta} = \vec{\theta} - \alpha(\vec{\theta}) \tag{2.2}$$

This is known as the **lens equation** and it is a fundamental equation in gravitational lensing theory.

Assuming the lens does have circular symmetry, general relativity tells us that a light ray in Figure 2.2 will be deflected by an angle $\hat{\alpha}$ such that:

$$\hat{\alpha} = \frac{4GM}{c^2\xi} \tag{2.3}$$

where M is the mass of the (point) deflector (see, for example, Schneider 1992).

The small angle approximation means this deflection angle may be related to the observed shift

α as:

$$\alpha = \frac{D_{LS}}{D_S}\hat{\alpha} \tag{2.4}$$

which yields an expression for the visible deflection by the lens of:

$$\alpha = \frac{D_{LS}}{D_S}\frac{4GM}{c^2\xi} \tag{2.5}$$

Substituting this into the (scalar) lens equation, we have:

$$\beta = \theta - \alpha = \theta - \frac{D_{LS}}{D_S}\frac{4GM}{c^2\xi} \tag{2.6}$$

and since $D_L\theta = \xi$,

$$\implies \beta = \theta - \frac{D_{LS}}{D_L D_S}\frac{4GM}{c^2\theta} \tag{2.7}$$

From this expression, it can immediately be seen that if the source S lies directly behind the lens L, then $\beta = 0$ and we will have:

$$\theta_E \equiv \theta = \sqrt{\frac{4GM}{c^2}\frac{D_{LS}}{D_L D_S}} \tag{2.8}$$

In such a situation, the light from a source will be smeared into a circle of radius θ_E, known as an *Einstein ring* (see, for example, Figure 1.2). The angular size θ_E is known as the *Einstein radius*, and it is important to note that it is a function only of the source redshift, the lens redshift, and the lensing mass M; it is not an intrinsic property of the lens alone. Typically, the value of θ_E for galaxy-galaxy lensing is of the order of an arcsecond, whereas lensing by a cluster of galaxies leads to a value about 10 times bigger. These are particularly important concepts in gravitational lensing, and will be referred to throughout this project.

Substituting expression 2.8 into 2.7, we further find that:

$$\beta = \theta - \frac{\theta_E^2}{\theta} \tag{2.9}$$

to yield a quadratic equation, the two solutions for which are:

$$\theta_{1,2} = \frac{1}{2}\left(\beta \pm \sqrt{\beta^2 + 4\theta_E^2}\right) \tag{2.10}$$

From expressions 2.8 and 2.10, we therefore learn that for lenses that are point masses, we obtain either an Einstein ring if $\beta = 0$, or exactly two images if $\beta \neq 0$. In the latter case, the two images are on either side of the source, with one image inside the Einstein ring and the other outside it. As the source moves away from the lens (that is, for β increasing), one of the images will approach the lens (and become fainter), while the other image approaches the true position of the source. In reality, lenses are usually extended and lumpy objects (a cluster of galaxies, for example) so there are likely to be more than two images.

A notable feature of gravitational lensing is that it preserves surface brightness. This is a consequence of Liouville's Theorem, and the absence of emission and absorption of photons in gravitational light deflection (Misner et al. 1973). However, it will change the apparent solid angle of a source. Consequently, the total flux received from a gravitationally lensed image of a source will be changed in proportion to the ratio of the solid angle of the image to that of the source. This results in a magnification of the source image:

$$\text{magnification} = \frac{\text{image area}}{\text{source area}}.$$

In terms of Figure 2.2, the magnification factor μ for a circularly symmetric lens can therefore be written as:

$$\mu = \frac{\theta}{\beta}\frac{d\theta}{d\beta} \tag{2.11}$$

(where, for the special case of a point mass, it can readily be shown that $\mu > 1$). Typically, when the source position β is around θ_E or less, the magnification will be strong. On the other hand, if $\beta >> \theta_E$, there is likely to be very little magnification.

This expression may be generalised for the case of a lens that is not circularly symmetric. The properties of a lens mapping are described by its Jacobian matrix A, namely:

$$A \equiv \frac{\delta\vec{\beta}}{\delta\vec{\theta}} = \left(\delta_{ij} - \frac{\delta\alpha_i(\vec{\theta})}{\delta\theta_j}\right) \tag{2.12}$$

(where δ_{ij} is the Kronecker delta).

The matrix A is called the *inverse magnification tensor*, and since the solid angle element $\delta\beta^2$ of the source is mapped to the solid angle element of the image $\delta\theta^2$, a generalized expression for the magnification becomes:

$$\frac{\delta\theta^2}{\delta\beta^2} = \frac{1}{detA}. \tag{2.13}$$

In addition to magnification, gravitational lensing will also introduce an element of distortion to the image. To describe this, it is helpful to rewrite the tensor A in terms of its eigenvalues, the usual form of which is:

$$A = (1-\kappa)\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \gamma \begin{bmatrix} cos2\phi & sin2\phi \\ sin2\phi & -cos2\phi \end{bmatrix} \tag{2.14}$$

(For a more detailed derivation of this expression see, for example, Serjeant 2010, Narayan & Bartelmann 1996)

The first term with κ produces an isotropic expansion: acting alone it maps the source onto an image with the same shape but different size. The term κ is known as the *convergence*. The γ term is known as the *shear*, and describes the magnitude of anisotropy or astigmatism, whereby the image shape is stretched in the ϕ direction (and shrunk in the perpendicular direction); see Figure 2.3.

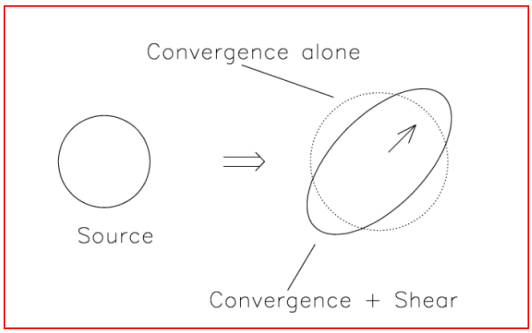

Figure 2.3: Convergence & Shear
(adapted from Narayan & Bartelmann 1996)

The eigenvalues of A are $1 - \kappa \pm \gamma$; for a circular source of unit radius, they correspond to an elliptical image with major and minor axes given respectively by:

$$\mu_t = (1 - \kappa - \gamma)^{-1}, \qquad \mu_r = (1 - \kappa + \gamma)^{-1} \tag{2.15}$$

In other words, μ_t and μ_r measure the amplification in the tangential and radial directions respectively, with the magnification μ given by:

$$\mu = \frac{1}{detA} = \frac{1}{(1-\kappa)^2 - \gamma^2}. \tag{2.16}$$

It can be seen that the values of μ_t and μ_r are infinite in those cases where the eigenvalues are zero. These define the two curves in the lens plane known respectively as the tangential and radial *critical curves*: an image forming along the tangential critical curve will be strongly distorted tangentially to that curve, whereas an image close to the radial curve will be stretched in a direction perpendicular to that curve. When critical curves are mapped onto the source plane, they define source positions known as *caustics*: a source lying on a caustic therefore gets infinitely magnified. (Strictly speaking, the real effect will be finite due to the actual wave nature of light, although in practice the fact that sources are not *point* sources is more important for ensuring a finite solution (Ellis et al. 2012)).

An example for the case of an *extended* source lensed by a symmetric lens is illustrated in Figure 2.4. On the left side of the figure, the outer and inner curves represent the tangential and radial critical curves respectively. A source close to the caustic at the lens centre, shown on the right side, produces the two arc images close to the tangential critical curve, whereas a source on the outer caustic produces both the elongated image on the radial critical curve and the elongated image outside the tangential critical curve.

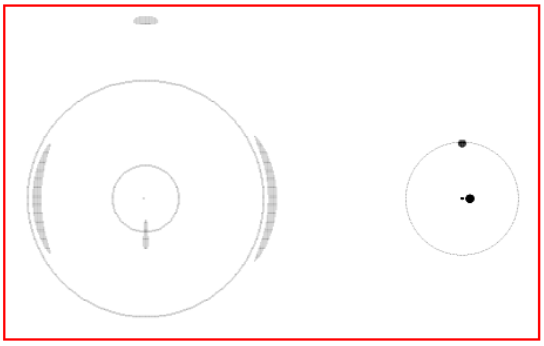

Figure 2.4: Critical Curves (left) & Caustics (right)
(adapted from Narayan & Bartelmann 1996)

So far we have (implicitly, if not always explicitly) been referring to gravitational lensing by a *point* mass. Clearly, this is not a realistic representation of a galaxy. However, many galaxies are observed to have a fairly flat rotation curve, which means that along much of their radius, the velocity dispersion σ_v is independent of the distance r from the centre; equivalently, such galaxy lenses have a slow 'fall-off' of deflection angle with r. A better approximation would therefore be to treat galaxies as though they were particles in an ideal gas, confined by a spherically symmetric gravitational potential. The gas is taken to be in thermal and hydrostatic equilibrium, and subject to the ideal gas law

$$p = \frac{\rho k T}{m} \tag{2.17}$$

where p is pressure, ρ is density, m is galaxy mass, T is temperature, and k is the Boltzmann constant. For such galaxies, it turns out (e.g. Meneghetti 2006) that the density ρ can be related

to the velocity dispersion σ_v as:

$$\rho(r) = \frac{\sigma_v^2}{2\pi G}\frac{1}{r^2} \tag{2.18}$$

Referring to Figure 2.2, we can then derive the *surface mass density* Σ as:

$$\Sigma(\xi) = \frac{\sigma_v^2}{2G}\frac{1}{\xi}. \tag{2.19}$$

This mass distribution is known as a *singular isothermal sphere* (SIS); the density profile is singular because, in the absence of any modifications, the mass density and the surface density tend to infinity as r and ξ respectively tend to zero.

For an SIS, the total mass enclosed within a projected distance ξ is given by:

$$M(\xi) = \int_0^{\xi} \Sigma(\xi')2\pi\xi' d\xi' = \frac{\pi\sigma_v^2}{G}\xi. \tag{2.20}$$

Birkhoff's theorem tells us that the deflection is dependent only on the mass contained within the Einstein radius, so this can be related back to the Einstein radius θ_E by analogy to expression 2.8, to give:

$$\theta_E = \sqrt{\frac{4GM(\theta_E)}{c^2}\frac{D_{LS}}{D_L D_S}} \tag{2.21}$$

hence,

$$\theta_E^2 = \frac{4GM(\theta_E)}{c^2}\frac{D_{LS}}{D_L D_S} = \frac{4G}{c^2}\frac{\pi\sigma_v^2\xi}{G}\frac{D_{LS}}{D_S}\frac{\theta_E}{\xi} \tag{2.22}$$

from which finally,

$$\theta_E = \frac{4\pi\sigma_v^2}{c^2}\frac{D_{LS}}{D_S} \tag{2.23}$$

This is an important relationship and corresponds to expression (2) in Collett (2015), where an SIS is assumed within the model. Furthermore, for an SIS strong lens, counter-images are separated by twice the Einstein radius, and magnification is purely tangential (Schneider 1992). For counter-images to be resolved therefore, it is a requirement of Collett's model that the quadrature of the sum of the seeing and twice the source size must be less than twice the Einstein

radius, and that for the tangential shearing to be observable the product of the source size and the magnification must be greater than the seeing; these criteria correspond to expressions (7) and (8) respectively in Collett (2015). These features will be addressed further in forthcoming sections.

2.3 Discussion - Applications of Gravitational Lensing

The applications of gravitational lensing fall broadly into three categories, which follow naturally from the theory described above (Narayan & Bartelmann 1996). Firstly, the magnification effect allows the identification of objects that are either too distant or too intrinsically faint to be observed without lensing. This idea was expressed some 80 years ago in an article by Zwicky (1937), who suggested that clusters of galaxies could be used as natural telescopes to search for magnified images of very distant galaxies. It took another 60 years however before any real observational progress was made, one of the limiting factors being the requirement for a background source to be substantially more distant than the lens: it was not until the quasar surveys in the 1970s that distant sources could be revealed. Gravitational lenses afford a level of resolution or sensitivity far higher than current direct observational limits. Earlier this year, the Hubble Space Telescope observed the furthest star ever seen when its brightness was momentarily magnified some 2,000 times, as a result of lensing by a foreground galaxy cluster; at a redshift of 1.49, the star would otherwise have been too faint to observe[1].

Secondly, gravitational lensing is independent of the composition or luminosity of a lens, relying solely on its two-dimensional mass distribution. This makes gravitational lensing ideal for the detection of dark matter, and for the study of its distribution and properties; see, for example, Massey et al. (2010). We may note further that the *wavelength* of light is not affected by gravitational lensing, so a test for lensing is that candidate images should possess the same spectral features (including redshift), although strictly speaking, for an extended source there may be differences in the observed spectra due to differential magnification; environmental factors, such

[1]See, for example, *https://www.spacetelescope.org/news/heic1807*

as absorption of light by dust around the lensing galaxy, may also need to be taken into account.

Lastly, the properties of individual lens systems are dependent on the geometry of the universe. As such, they represent potential constraints on fundamental cosmological parameters, including those governing the evolution of the dark energy equation of state; see, for example, Grillo et al. (2008) and Golse et al. (2002). The effectiveness of strong gravitational lensing in this regard is a key issue of this work and will be the subject of a later chapter.

In conclusion to this section, and by way of 'setting out my stall', this thesis involves making predictions for forthcoming strong gravitational lens surveys, with a view to their application for constraining fundamental cosmological parameters, and to quantifying the scale of data anticipated for strong lensing. In this chapter 2, I have outlined the theory behind gravitational lensing, and in chapter 3 the structure of the model (and its modification) will be discussed - with its predictions for forthcoming surveys detailed in chapter 4. In chapter 5, I discuss the extent to which the Euclid strong lensing survey can be used to constrain cosmological parameters, and in chapter 6, consideration will be given to an application of the model to submillimetre galaxies. Finally, in chapter 7, the results will be discussed and options for further work presented.

Chapter 3

The Model

Abstract

This chapter commences with a high level outline of the model designed by Collett (2015) for predicting the number and properties of galaxy-galaxy strong lenses discoverable in forthcoming surveys, the aim being to provide a helpful overview of its methodology. More rigorous descriptions, and analyses, of the code will follow in subsequent sections of the project or, where otherwise more appropriate, in the Appendix. Following the outline, I present an 'audit' of the code. Here I discuss those areas in the original Python[1] scripts of the model where I believe there are potential discrepancies, either within the code itself or between the code and the text of the article by Collett (2015). Whilst a number of issues have been identified, there is nothing that substantially affects the results obtained from initial applications of the model to surveys by Euclid: with some notable exceptions, where discrepancies have been confirmed, the corresponding corrections have tended to be minor. It should be borne in mind throughout that certain key assumptions behind the original model, as well as the initial results, are detailed in Collett (2015). Some of these assumptions, in particular those that relate to galaxy evolution and luminosity functions, are addressed in a later section of this project where the model's sensitivity to these will be considered.

[1]Python v2.7 has been used by me throughout this project; with the exception of instances where scripts have been run on the OU cluster, this version of Python has been provided as part of the Canopy (v1.7.4) package retrieved from *http://www.enthought.com*.

3.1 Methodology

One of the objectives of the model, as originally developed, was to predict the number and nature of galaxy-galaxy strong lenses that would be detected by the forthcoming Euclid surveys. For the purpose of this prediction, the population of background galaxies (representing the potential sources) is based on data from the sky catalogues simulated for the LSST by Connolly et al. (2010). It should further be noted that in the code available for the model on *GitHub*, on which this project is based, the survey parameters are set *by default* to those of the Euclid Wide Field survey.

As a first step in undertaking this project, it was essential to produce a map of the key dependencies within the code in order to understand sufficiently the model's methodology; this mapping is provided in Appendix A.2. However, that level of detail is not strictly necessary to appreciate the results or conclusions of my work. Instead, a more useful description of the methodology is best described by reference to the three consecutive stages outlined below.

Stage One: Creating an Idealised Lens Population

The first stage of the code - executed by the module *MakeLensPop* - proceeds as follows:-

- Creates an idealised set of properties for foreground galaxies in the range z=0 to z=2; these will be potential lenses (or *deflectors*).
- Uses LSST simulated data to provide a set of properties for background galaxies up to z=10; these will be potential sources.
- To the properties of each potential source, chosen at random, appends the properties of a potential lens (takes the first source and appends the first lens, the second source and the second lens, the third source and the third lens, and so on); the potential sources are sampled as many times as there are potential lenses.
- For each of these pairings, calculates the Einstein radius.
- For each of these pairings, obtains a random pair of coordinates for the location of the

(centre of the) source with the lens defined as the origin. The coordinates are chosen from a range that reflects the source density of the survey.

- Determines if the location of the source is within the Einstein radius. If so, treats the pairing as an idealised lens system; this corresponds to expression (6) in Collett (2015).

Stage Two: Identifying Detectable Lens Systems

The second stage of the code - executed by the module *ModelAll* - continues as follows:-

- Using the *seeing* value of the survey, and the lens-source properties, checks that the quadrature sum of the *seeing* and twice the source size are less than twice the Einstein radius; this corresponds to expression (7) in Collett (2015).
- Using the magnification, *seeing* and Signal-to-Noise values, checks compliance with the other criteria required for detection by Euclid; these correspond to expressions (8) and (9) in Collett (2015). For evaluating their respective pixels, the *lensed source* galaxy and the *lensing* galaxy are placed in a 200x200 'postage stamp'; the pixels for the *unlensed* source galaxy are evaluated using a 50x50 'postage stamp'.
- If the above criteria are met, classifies the lens-source pair as detectable by Euclid.

Stage Three: Properties of Detectable Lens Systems

The final stage of the code - executed by the module *MakeResults* - concludes as follows:-

- Scales up the results of the previous stage to take into account the area covered by the survey and the fraction of the sky used by the code in the previous stages.
- Stores the results - namely, the properties of the detectable lens systems - in a worksheet (*lenses_Euclid.txt*) that is then available for further analysis.

3.2 Audit of the Code

In this section, I discuss discrepancies that I have identified either within the code itself, or between the code and the text of the article by Collett (2015). In each case, the corresponding lines

of the source code are shown[2] together with a brief description of the issue and its implication for the model; a comment has also been added where some elaboration has been felt necessary. For ease of readability, the areas have been grouped according to the methodology described in section 3.1.

A key to the abbreviations used for the Python modules referred to in this and subsequent sections of the project is given in Table 3.1. The corresponding source codes themselves may be found in Appendix A.

Table 3.1: Module Abbreviations

Abbreviation	Module
MLP	MakeLensPop.py
Dis	Distances.py
PFs	PopulationFunctions.py
MAll	ModelAll.py
FLS	FastLensSim.py
SBM	SBModels.py
SBP	SBProfiles.py
Sur	Surveys.py
Sto	StochasticObserving.py
SN	SignaltoNoise.py
MRs	MakeResults.py

Stage One: Creating an Idealised Lens Population

3.2.1 Creating Lens Population Splines

Lines: 247→8→17 MLP→156 PFs

Description: The initialisation routine for the *LensPopulation* object class includes the *beginLensPopulation* function, which chiefly provides lens number density and velocity dispersion splines against redshift. The routine in lines 159-174 (PFs) is not executed as *reset* is hardcoded

[2] A '→' symbol is intended to clarify the order in which lines are executed, e.g. $10 \rightarrow 5 \rightarrow 20$ indicates that the routine flows from line 10 to line 5 to line 20.

as True.

Comment: If *reset* were not set to True, then the code would load in splines from the existing *lenspopsplines* pkl file. With *reset* set to True, splines are created by the *Psigspline* and *Colourspline* functions, and written to a new *lenspopsplines* pkl file by the *lensPopSplineDump* function.

Implication: Inefficient use of computer time, as the procedure for creating and storing data (splines) is duplicated.

3.2.2 Source Density Function I

Line: 195 PFs

Description: For consistency with expression (3) in Collett (2015), the term *dphisiggivenz* should read *dndsiggivenz*.

Implication: Minimal - code readability only.

3.2.3 Source Density Function II

Line: 196 PFs

Description: As above, for consistency with Collett (2015), the term *phisigspline* should read *nsigspline*.

Implication: Minimal - code readability only.

3.2.4 Lens Sample Wrapper - Lens Population Splines

Lines: 249→68→8→17 MLP→ 156-176 PFs

Description: The difference between the execution of the *beginLensPopulation* function following line 247 (MLP) and following line 249→68 (MLP) is that *reset* = True and *reset* = False respectively; but in line 157 (PFs) *reset* is overridden as True.

Comment: The initialisation routine for the *LensSample* class object includes creating an object of class *LensPopulation*, and the initialisation routine of the latter includes the *beginLensPopulation* function. As described earlier, if *reset* were left set as False, then the *beginLensPopulation* function would load in splines from the existing *lenspopspline* pkl file, whereas setting *reset* = True means that splines must be created by the *Psigspline* and *Coloursplie* functions, and then written to a new *lenspopsplines* pkl file by the *lensPopSplineDump* function.

Implication: Inefficient use of computer time, as the procedure for creating and storing data is duplicated by *reset* having been hardcoded as True.

3.2.5 Lens Sample Wrapper - Redshift Splines

Lines: 249→68→8→16 MLP→12-29 PFs

Description: There seems to be no difference in parameters between the function *beginRedshift-DependentRelation* that is executed following lines 249 → 68 → 16 (MLP) and following lines 72 (MLP) → 104 (PFs.). This routine is designed to create (or store) distance and volume splines against redshift.

Comment: Initialisation of class object *LensSample* includes the creation of a new class object *EinsteinRadiusTools*, and initialisation of that routine includes the function *beginRedshiftDependentRelation*. But that routine has already been run previously with the same parameters as part of the initialisation routine for the class object *LensPopulation*. Note that both class objects *LensPopulation_* and *LensPopulation* have initialisation procedures which respectively call *beginRedshiftDependentRelations* functions.

Implication: Inefficient use of computer time, as routines are duplicated (although at least setting *reset* = False prevents duplication of the procedures for deriving and storing spline data).

3.2.6 Number of Deflectors

Lines: 286-291 PFs

Description: This routine integrates the density function to return the number of potential lenses (or *deflectors*) in a given redshift interval. The function used is dependent on the cosmology adopted, via the implications of the cosmological parameters on the comoving volume (see lines 202 & 211 PFs).

Comment: Arguably, the model should assume a *fixed* number of potential deflectors, independent of the cosmology adopted. This issue became apparent only during later runs of the code whilst investigating the subject of cosmological constraints for this project; further details, including the corresponding modifications made at that time, are discussed in section 5.2.2.

Implication: Unless corrected, extreme values for the cosmological parameters will have a significant effect on the predicted number of idealised lenses (e.g. setting $\Omega_m = 0.9$ results in a reduction of about 50% compared to the standard cosmology), and therefore any results in respect of surveys under such cosmologies will not be reliable.

3.2.7 Flattening Parameter

Line: 310 PFs

Description: This routine uses the model given in expression (4) of Collett (2015) to derive the flattening parameter of an elliptical galaxy. However, there is a discrepancy between the x coefficient in the formula in the code compared to expression (4) in the article.

Comment: According to the article, the σ coefficient and the constant term should be 0.38 and

0.00057 respectively. But in the code, the σ coefficient is given as -0.00057. *This has subsequently been confirmed in private correspondence with Dr Collett to be a typographical error in the article and not an error in the code.*

Implication: This would give an inaccurate relationship between the velocity dispersion of a galaxy σ and the flattening parameter q.

3.2.8 Source Light Profiles

Lines: 423-426 PFs

Description: The code here is intended to return a value for the effective radius for a given light profile. However, the effective radius (*r_phys*) values derived separately in lines 423 and 426 are not the same (as claimed in the module narrative), and the formula initially appears to be wrong in any case.

Comment: The *RofMz* function returns the value for the effective radius of a galaxy based on its Magnitude M and redshift z. The relation is given in expression (5) of Collett (2015). The code expresses the effective radius *r_phys* as:

$$r\_phys = 10^{R} \times \frac{(1+z)}{1.6}^{-1.2}$$

where $R = \frac{-(M+18)}{4}$

It is not clear how this relates to the expression in Collett (2015), and it appears to be very different. Furthermore, the code in lines 423 is overwritten by the code in lines 425 and 426; although lines 425 and 426 do not match the expression in Collett (2015) either, taken together they do appear to represent a plausible and acceptable expression for the effective radius.

Implication: This could prove of significance given the importance of the effective radius as a galaxy property, although as stated above it seems the code ultimately used in the program may

be acceptable after all. *This issue is addressed in further detail in Appendix C.*

3.2.9 Number of Sources per Lens

Lines: 124/126 MLP

Description: The value of *nsources* represents the number of sources pertaining to any one lens. This parameter may be varied manually, which suggests an option to allow for more than one source, but the code does not appear to accommodate this correctly.

Comment: The *Lens?* field depends only on the *first* background galaxy data appended to the lens record, so if *nsources* is varied manually such that $nsources > 1$, this field may be flagged as True even if subsequent background galaxy data appended to it do not correspond to sources.

Implication: There are likely to be errors if the model is applied to lens systems with multiple sources. The model used in this project however is restricted to single source lenses, so this issue may be ignored for the purpose of this project.

3.2.10 Storing Source Redshift

Lines: 125/150 MLP

Description: There is a duplication here.

Implication: Trivial.

3.2.11 Storing Einstein Radius

Lines: 126/151 MLP

Description: There is a duplication here.

Implication: Trivial.

3.2.12 Storing Halo & Stellar Masses

Lines: 157/158 MLP

Description: The data set loaded in as part of the initialisation routine for the class object *SourcePopulation* includes data for halo and stellar masses (originally *mhalo* and *mstar* respectively). However, the halo mass and stellar mass elements become transposed at this stage: the value of the halo mass is therefore stored in a data field called *mstar*, and the value for the stellar mass is stored in a field called *mhalo*.

Implication: These values are not used elsewhere in the code. In any case, they are unlikely to be of significance since the mass of a source object does not feature in the lensing criteria or formulae. Note there is an incorrect reference in Collett (2015) to 'the density profile of the source': this should read 'the density profile of the *lens*'.

Stage Two: Identifying Detectable Lens Systems

3.2.13 Sample Size

Lines: 73-79 MAll

Description: A sample of just one tenth of the number of idealised lenses is evaluated against the Euclid detection criteria for this stage of the model, but the number of idealised lenses has been *hardcoded* as 12,530,000.

Comment: Sampling a *fixed* number of idealised lenses (ie. 1,253,000) means the *fraction* of idealised lenses sampled for applying the Euclid detection criteria (compared to the total number of idealised lenses) will differ according to the cosmology applied. Yet the code ultimately

scales up by a factor of ten regardless of the actual fraction. The rationale behind choosing a *fraction* of the sky is driven by runtime practicalities; a sample of 1.2 million idealised lenses takes approximately 10 hours.

Implication: After evaluating how many lens systems pass the Euclid criteria, that number is scaled up eventually within the code by a factor of ten to return a number that Euclid would detect if it were faced with *all* the idealised lenses, although it is then reduced again to the extent that the survey area is roughly $\frac{15{,}000}{42{,}000}$ of the whole sky (see also lines 116, 169, 189 MRs). There are approximately 11.9 million idealised lenses in the standard (or Concordance) cosmology, so scaling up by ten is acceptable given the true scaling factor should be $\frac{11{,}900}{1{,}253} = 9.5$. However, under different cosmologies the number of idealised lenses may vary considerably, so in those cases scaling up by ten may not be accurate and the lens predictions would be unreliable; as it happens, after correcting for the issue in section 3.2.6, the model predicts a similar number of idealised lenses when tested throughout the Ω_m range in section 5.2.2, so for our purposes the scaling may be acceptable after all.

3.2.14 Initialising Magnification & Source Magnitudes

Lines: 110/113 MAll

Description: There is a duplication of *lenspars[mag]* and *lenspars[msrc]* in the initialisation routine for these variables.

Implication: Trivial.

3.2.15 Sersic Profile Attributes

Lines: 29-32 & 33/54 SBM

Description: The *value* attribute is not valid here, so the exception routine *__setattr__* is always executed; also *vmap* is empty, so the *setPars* function does not run.

Comment: The attribute *.value* does not appear to be a valid term here. This means the exception routine will always run and the dictionary *vmap* is not populated. Hence also the *setPars* function, which relies on a *for key in vmap* loop, does not execute.

Implication: This is not of any obvious significance within the model.

3.2.16 Einstein Radius & Seeing Criteria I

Lines: 139 Sto

Description: This corresponds to expression (7) in Collett (2015), but there is a discrepancy between the code and the article text.

Comment: The criteria used in the code is

$$2\theta^2 < (2r)^2 + s^2$$

whereas the criteria given in the article is

$$4\theta^2 < (2r)^2 + s^2$$

This results in a difference of factor 2.

Implication: This will have an effect on the predictions of the model, as it will be reflected in the level of resolution required for the detection of lenses.

3.2.17 Setting Axes & Distance Array

Line: 56 SBP

Description: The x, y coordinates have been transposed in the distance formula compared to

expression (1) in Collett (2015).

Comment: Transposing the x, y coordinates could imply the axes are no longer aligned with the semi-major and semi-minor axes used for defining the flattening q, on which other properties of the galaxy are based.

Implication: This is unlikely to be of significance, given the stochastic nature of the x, y coordinates.

3.2.18 Determining Magnification Values

Lines: 168-176 FLS

Description: The use of *sum* in the definition of *unlensedsrcmodel* and the definition of *srcnorm* as the **sum of** *unlensedsrcmodel* elements means that the subsequent definition of *unlensedsrcmodel* = *unlensedsrcmodel/srcnorm* = 1 always.

Implication: This is not of any significant concern, other than one of readability: the *unlensedsrcmodel* component in line 176 does not require the *sum* operation on it, as it is always equal to one. (Note that *srcmodel* is already normalised on *srcnorm*, so the magnification calculation is valid).

3.2.19 Einstein Radius & Seeing Criteria II

Lines: 98-99 SN

Description: This corresponds to expression (7) in Collett (2015) and implies a potential discrepancy of factor 2.

Implication: *as above for line 139 (Sto)*

3.2.20 Pyfits Module

Line: 3 MRs

Description: The *pyfits* module is not used and this line may be commented out.

Implication: Trivial.

3.2.21 Assigning Co-Add Descriptions

Lines: 51-56 MRs

Description: This routine checks the element in *survey[-2]* for 'a', 'b', or 'c' and replaces that character with the corresponding co-add description to return a new variable called *surveyname*. However, the wrong position has been used for the element extraction: position [-2] is incorrect and should instead read [-1].

Implication: Trivial. The variable *surveyname* does not appear to be used subsequently, and may be commented out.

3.2.22 Initialisation of Source Radius

Lines: 76/82 MRs

Description: There is a duplication of *rs[key]* in the initialisation routine for this variable.

Implication: Trivial.

3.2.23 Storing Lens Radius

Lines: 132/135 MRs

Description: This routine writes the values of the key parameters of each lens system to a text file, but there is a duplication of the Write instruction for the element corresponding to the radius of the lens *rl*.

Comment: The parameter *rl* occurs twice in the *lenses_[survey].txt* file.

Implication: The (second, ie. duplicated) position of the parameter is not consistent with the narrative in the example text file from *GitHub*, so any analysis relying on this file may inadvertently misinterpret this value. The number and details of the lenses detectable are otherwise unaffected.

3.2.24 Storing Source Magnitude

Lines: 140/141 MRs

Description: In the same routine as 3.2.23 above, the element corresponding to the magnitude of the source *ms* is **omitted** from the Write instructions.

Comment: The parameter *ms* is not included in the *lenses_[survey].txt* file.

Implication: This does not affect the number or details of lenses detected, but any subsequent analysis carried out using this text file may be compromised: *ms* is **not** in position [12] of the text file as might have been expected from the example text file in *GitHub*.

3.2.25 Assigning Co-Add Descriptions

Lines: 203-210 MRs

Description: *as above for lines 51-56 (MRs)*

3.3 Discussion - The Modified Model

In this chapter, we have discussed the objectives and methodology of the model as designed by Collett (2015). A review was carried out to investigate any potential errors or inconsistencies in the code and, where appropriate, modifications made to resolve them. Inevitably with a program of this size (approximately 3,000 lines of Python code), some 'bugs' were found to be present but most of these were minor. Consequently, the amendments do not result in any significant changes to the predictions of the model in its *default* mode, which assumes a standard ΛCDM cosmology: for example, the prediction by the modified model of around 180,000 lenses discoverable by Euclid is within 10% of the 166,000 lenses predicted by the original model.

Some of the more significant discrepancies are those that affect the model when it comes to exploring its application under different cosmologies. One of these concerns the use of a scaling factor which relies on a value for the number of idealised lenses that has been hardcoded in the model, but which can in fact vary by as much as 10% under different values of the cosmological parameters (see 3.2.13). Likewise, there is also an issue with the model's estimate for the number of potential deflectors, which again becomes relevant when running the model under different cosmologies. In this case, the model should be assuming a fixed number of deflectors, regardless of cosmology, but is instead allowing this value to vary through changes to the comoving volume (see 3.2.6); this is addressed in more detail (and a correction discussed) in a later chapter.

Another 'bug' worthy of note is the confusion of the *source galaxy magnitude* with the *lens galaxy radius* in the data exported to a *txt* file by the original model for the purpose of analysis. A key to the fields and their contents in that file is provided in the documentation for the model, so the omission of the former galaxy property and the duplication of the latter galaxy property in the contents of the file would prove problematic for any interpretation that relies on the integrity of that description (see 3.2.24).

An inconsistency between the coding of the model and the description by Collett (2015) was also identified in connection with the light profile of a source (see 3.2.8). Whilst on the face of it the discrepancy is a significant one, and it is the subject of Appendix C, it turns out that the implications of this inconsistency are minor since an acceptable expression for the light profile is applied ultimately in the model.

The predictions of the modified model, together with an application of the model to the *Deep Field* survey by Euclid, are discussed further in the next chapter.

Finally, this chapter serves, amongst other things, to illustrate the value of open source programming. The ability to download, run, analyse and, where appropriate, amend existing scripts clearly affords an opportunity to draw on resources beyond those likely to be available to any individual or group acting in isolation. This sharing of software (and tests of the software) has come to play an increasingly significant role in space science, and is an aspiration of the new ESCAPE project[3] in which the Open University is a partner. *GitHub*, through which Collett's code is distributed, is an example of this - as is the availability of the Python-related modules used within this project, such as **astropy**.

Open source programming is part of the wider notion of open access research, which is the subject of a number of recent initiatives to facilitate access by researchers to resources and data worldwide. An example of one such initiative is the European Open Science Cloud (EOSC), the launch of which was announced by the European Commission, following the adoption of the Digital Single Markets strategy on 6 May 2015. The stated aim of this project is "to create a trusted environment for hosting and processing research data to support EU science in its global leading role". It is anticipated that EOSC will provide 1.7m EU researchers with an environment that has free, open services for data storage, management, analysis and re-use across disciplines: by connecting existing and emerging horizontal and thematic data infrastructures, the intention is to bridge otherwise fragmented or ad-hoc solutions[4].

[3] *See https://indico.in2p3.fr/event/18279/.*
[4] See *https://ec.europa.eu/research/openscience/pdf/eosc_strategic_implementation_roadmap_short.pdf.*

Another initiative is the Open Universe program, proposed to the United Nations Committee on the Peaceful Uses of Outer Space (COPUOS). Recognising that internet technologies represent an unprecedented and extraordinary two-way channel of communication between producers and users of data, this initiative is intended to promote a dramatic increase in the availability and usability of space science data to 'extend the potential of scientific discovery to new participants worldwide'[5].

The Research Data Alliance (RDA) is an initiative that has sought to go beyond space science and cover a wider range of disciplines, such as agriculture, oceanography, climate and health. It was launched as a community-driven organization in 2013 by the European Commission, the US National Science Foundation and National Institute of Standards and Technology, and the Australian Governments Department of Innovation. The aim is to build a social and technical infrastructure to enable open sharing of data across barriers, through focused Working Groups and Interest Groups made up of experts from around the world in industry, academia and government. The RDA has over 7,000 members from 137 countries, with over 30 Working Groups and over 60 Interest Groups currently participating[6].

Major advances in technology over recent years have highlighted the significance of open access to research, and with examples such as those outlined above much progress has been made. This is a rapidly growing area however, and further efforts remain necessary both to consolidate and to expand the underlying services. The worldwide research community stands to benefit dramatically from the coordination and cooperation these opportunities will provide. In particular, the analyses discussed in this chapter illustrate perfectly the contribution of open-source software to research reliability and reproducibility.

[5]See *https://arxiv.org/abs/1805.08505*
[6]See *https://www.rd-alliance.org.*

Chapter 4

Predictions for Strong Lensing Surveys

Abstract

In this chapter, I consider the predictions for both the Wide Field and Deep Field surveys by Euclid, having modified the model in Collett (2015) to accommodate the issues raised in the previous chapter. I find that even after those modifications, the predictions for the Wide Field survey are not significantly different from those of the original model, with the total number of detectable lenses just under 10% higher. A number of adjustments are then made to the model in order to predict results for the Deep Field survey, from which it appears that an increased sensitivity of two magnitudes is largely responsible for the 7-fold increase in detectable lenses for that survey area. Further adjustments are subsequently applied to obtain predictions for the COSMOS and WFIRST missions. In the case of the former, comparisons with studies elsewhere indicate that a number of lenses remain to be confirmed in the survey area, whereas in the case of the latter, the number of discoverable lenses suggests the increased depth almost compensates for the smaller area when compared to Euclid's Wide Field survey. Finally, limitations of the model, including the types of lensing system accommodated and the assumptions behind the survey parameters, are outlined and their relevance discussed. One of the conclusions to be

drawn is that Euclid seems well-suited to high magnification events, and that the predictions of the model described in Collett (2015) are likely to underestimate the corresponding number of lensing systems discoverable by Euclid.

4.1 Euclid

The main differences between the Euclid Wide Field and Deep Field surveys are in their survey area and sensitivity: the Wide Field survey covers an area of approximately 20,000 sq.deg. with an integration time per pixel on the sky ('exposure time') of 1,610 secs., whereas the Deep Field survey covers an area of 40 sq.deg. with an exposure time of 64,095 secs. The details of the predictions are discussed below, but in summary the increased sensitivity of two magnitudes in the Deep Field survey broadly accounts for a 7-fold increase in the sky density of detectable lenses.

4.1.1 Euclid Wide Field

Predictions of the Original Model

The results of an initial application of the model are discussed in full in Collett (2015), but for ease of reference the results of my own run of the original model are summarised below in Table 4.1. For the avoidance of doubt, these results follow from the model *before* any modifications to resolve the discrepancies raised in section 3.2, and therefore serve as a 'consistency check' with the findings in Collett (2015).

In its original form, an application of the model resulted in a prediction of approximately 165,000 lenses detectable by Euclid.

Predictions of the Modified Model

Once the corrections were made, the model was run again to establish whether the modifications resulted in any significant changes to the predictions for the Euclid survey, compared to those of the original model. In the event, most of the amendments were minor, and consequently the

Table 4.1: Original Model Predictions

Parameter	Mean	Median	Variance
Lens redshift	0.71	0.64	0.13
Source redshift	1.94	1.83	0.76
Einstein radius (arcsec)	0.74	0.65	0.16
Velocity dispersion (km/s)	223	221	2393
Source magnitude	25.53	25.54	1.35
Magnification	7.59	5.47	40.3

results were not significantly affected. Key data from these predictions are shown below in Table 4.2, and illustrated in the histograms of Figure 4.1.

Application of the model in its modified form resulted in a prediction of 180,500 detectable lenses, just under 10% higher than the original prediction.

Table 4.2: Modified Model Predictions

Parameter	Mean	Median	Variance
Lens redshift	0.71	0.64	0.14
Source redshift	1.93	1.82	0.75
Einstein radius (arcsec)	0.72	0.63	0.16
Velocity dispersion (km/s)	220	218	2432
Source magnitude	25.45	25.47	1.41
Magnification	7.21	5.16	36.4

High Magnification Lensing

In addition to the above, the model was applied to investigate predictions for lensing systems of high magnification. Such systems are an example of a rare population that can only be discovered in large lensing surveys. They are useful for obtaining high resolution data on background galaxies, so predicting their existence is a worthwhile exercise. To this end, the population of detectable lenses was filtered to include only those where background sources were magnified by a factor of at least 10. The histograms in Figure 4.2 illustrate the distributions of their key properties, and the quantitative data is presented in Table 4.3. The model predicts that approximately 33,000 high magnification lensing systems would be detected by Euclid. In reality however, this is likely to be an underestimate, since in its present form the model does not

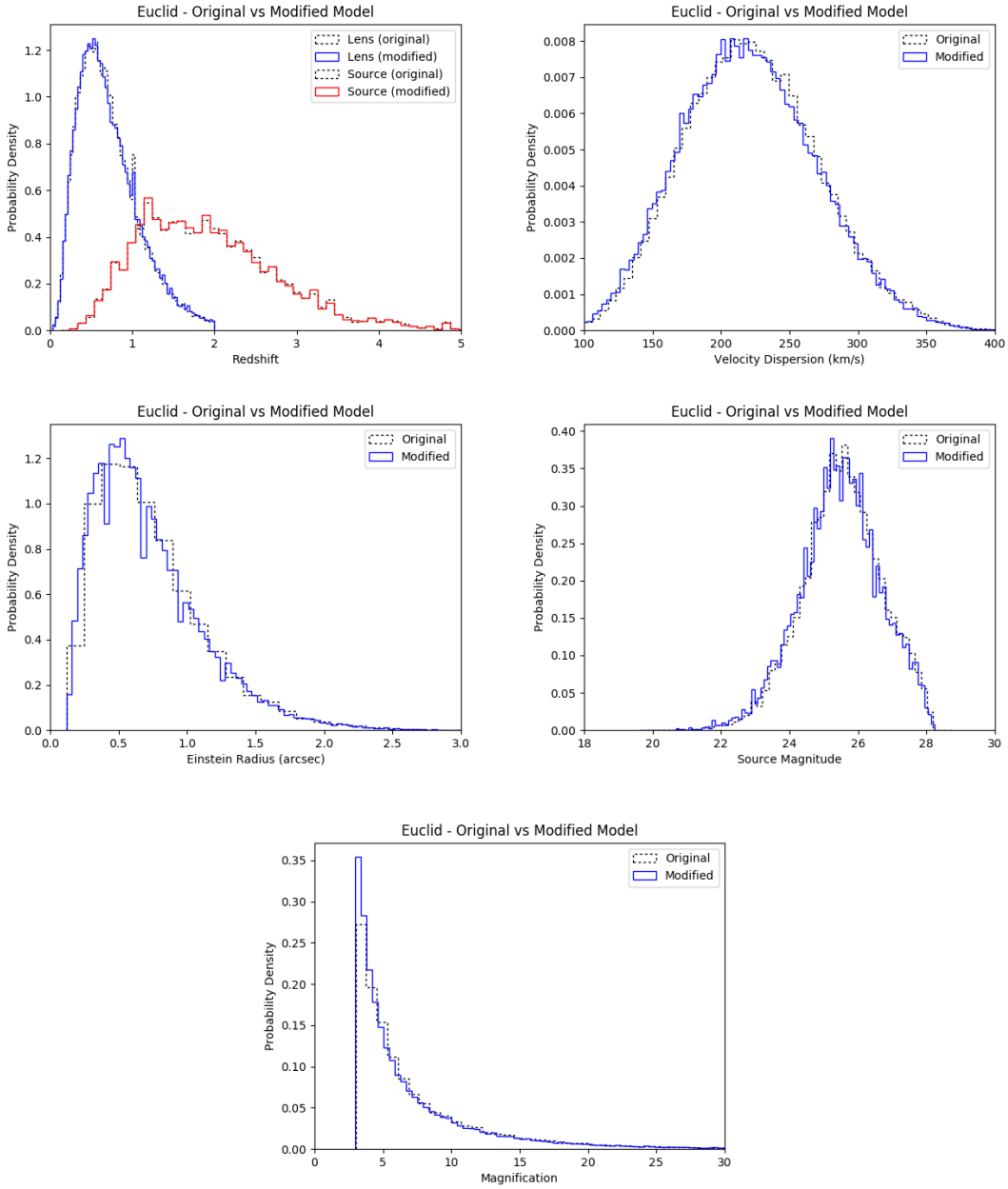

Figure 4.1: Properties of the lensing systems predicted by the modified model (solid lines) compared with the original predictions (dotted lines). Most of the modifications were minor and consequently the results are not significantly different, although there is an increase of approximately 10% in the number of lenses predicted.

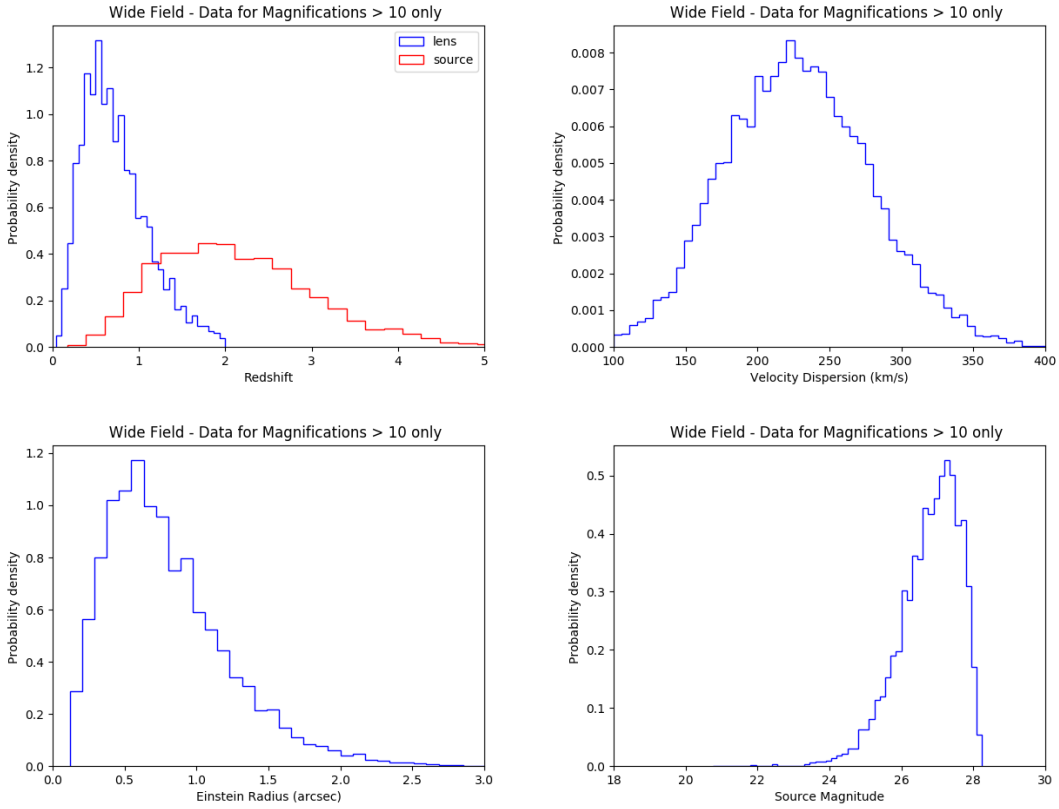

Figure 4.2: Properties of high magnification lensing systems predicted by the modified model. The plots relate only to lensing systems where sources have been magnified by a factor of at least 10. The prediction of 33,000 such systems is likely to be an underestimate, since the model does not take into account the existence of cluster lenses.

allow for cluster lenses, which are lenses that comprise *clusters* of galaxies rather than the single galaxies assumed in the code: such lenses could significantly increase the magnification of the associated sources.

4.1.2 Euclid Deep Field

In order to run the model for the Euclid Deep Field survey, a number of adjustments had to be made to the (otherwise corrected) code to reflect the difference in survey parameters compared to those used for the Wide Field survey.

Table 4.3: Modified Model Predictions - High Magnification Systems

Parameter	Mean	Median	Variance
Lens redshift	0.73	0.67	0.14
Source redshift	2.14	2.03	0.83
Einstein radius (arcsec)	0.80	0.71	0.18
Velocity dispersion (km/s)	228	227	2485
Source magnitude	26.77	26.9	0.72

Survey Parameters

The key module that stores the parameters corresponding to different surveys is *Surveys.py*, with the data expressed by way of properties of an object class *Survey()*.

The differences between the Euclid Wide Field and Deep Field surveys are reflected principally in the survey area and sensitivity, represented by the *degrees_of_survey* and *exposuretimes* parameters respectively. As mentioned previously, the default code in Collett (2015) relates to the Wide Field survey, with *degrees_of_survey* set for 20,000 sq.deg.; this is an approximation, with the Euclid Consortium Summary listing the survey area as 15,000 sq.deg. (We note the whole sky = 41,253 sq.deg., sometimes approximated to 42,000 sq.deg. within the code). The *exposuretimes* is set for 1,610 secs. The *degrees_of_survey* parameter is called by the *MakeResults.py* module to scale the results, after the *ModelAll.py* module has been run to analyse a fraction of the full-sky population of idealised lenses for their detectability. The *exposuretimes* parameter is called by the *StochasticObserving.py* module (within the *ModelAll.py* module) and is applied in the convolution procedure for the lens image.

According to the data in Laureijs et al. (2011), the code for the Deep Field survey needs to be adjusted for a survey area of 40 sq.deg. and an increased sensitivity of two magnitudes. The corresponding parameter values therefore requiring adjustment are shown in Table 4.4.

Table 4.4: Deep Field Survey Parameters

Parameter	Required Value	Location in Code
$degrees\_of\_survey$	40	line 127 Sur
$frac$	42000/40	line 94 MRs
$exposuretimes$	64095	line 123 Sur

Predictions

The number of detectable lenses predicted by the model for the Euclid Deep Field survey is $\sim 3,500$. This compares to $\sim 180,000$ for the Wide Field survey. Simply scaling the Wide Field survey to the Deep Field survey (ie. multiplying by $\frac{40}{15000}$) would only have resulted in a detection number of ~ 480. We may conclude therefore that the increased sensitivity of the Deep Field survey - namely, two magnitudes - is a significant factor behind the 7-fold increase in detectable lenses in the survey area.

Other key data of the predictions are shown in Table 4.5, and illustrated by the histograms in Figure 4.3.

Table 4.5: Deep Field Predictions

Parameter	Mean	Median	Variance
Lens redshift	0.76	0.69	0.15
Source redshift	2.25	2.17	0.81
Einstein radius (arcsec)	0.75	0.66	0.18
Velocity dispersion (km/s)	219	217	2443
Source magnitude	26.68	26.89	0.98
Magnification	5.32	4.27	11.1

4.2 COSMOS

During the course of my project, a colleague whose research involved investigating aspects of COSMOS asked whether the model in Collett (2015) could be applied to predict the number and nature of strong gravitational lenses that would be detected by that survey. This was not originally scheduled as part of my work, but since the model could indeed be readily modified to provide this information, it was felt both helpful and of academic interest to do so.

In this section, I outline the modifications made and the results obtained.

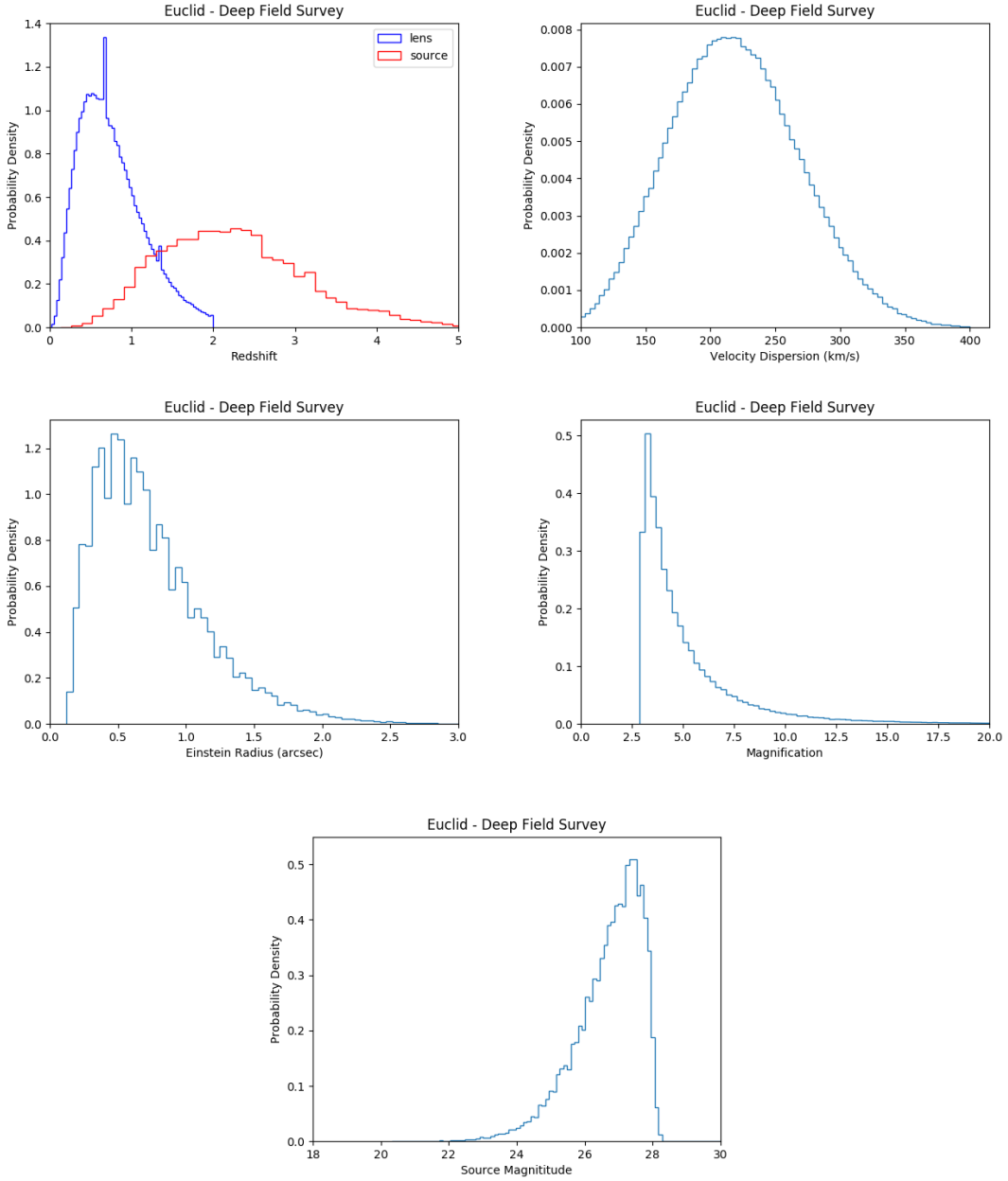

Figure 4.3: Properties of the lensing systems predicted for the Euclid Deep Field Survey. An increase in the sensitivity by two magnitudes is an important factor behind the 7-fold increase in detectable lenses (in the survey area) when compared to the Wide Field survey.

Survey Parameters

As already mentioned, the principal module for recording survey parameter data is *Surveys.py*. The survey parameters for COSMOS and their values are listed in Table 4.6.

Table 4.6: COSMOS Parameters

Parameter	Value
Survey area	1.7 sq. deg.
Filter band	F814W
Pixel size	0.049 (0.05) arcsec. per pixel
Side	204
PSF	0.085 arcsecs.
Zero exposure time	1 second
Zeropoints	25.9
Skybrightness	21.89 mags per sq. arcsec.
Exposuretime	8,937 seconds
Gains	2
Number of exposures	1
Readnoise	61.04 e

In addition, there is a parameter in the code called *stochasticobservingdata*, which is a *numpy* array comprising the seeing and skybrightness values, thus[1]:-

$$self.stochasticobservingdata = [twodF814]$$

$$twodF814 = numpy.array([[0.09, 21.9], [0.09, 21.9]])$$

Module Amendments

As well as the parameter changes listed above, amendments were made to three other modules to recognise both the COSMOS survey name and the F814W_ACS band filter; these are summarised as follows:-

FastLensSim.py (lines 297-298, 331): COSMOS survey name and F814W_ACS band name added.

MakeResults.py (lines 25, 33-34, 100-102, 154): COSMOS survey name added and 'scaling factor'

[1] See lines 7-10 Sto.

amended.

ModelAll.py (lines 11, 39-40, 175): COSMOS survey name added.

Predictions & Comment

There are 285,423 objects, nearly all of them galaxies, listed in the COSMOS field that are brighter than $I = 25$. In the study by Faure et al. (2008), a subset of 9,452 of these objects was chosen to be the most likely to contain gravitational lens systems. The findings of that study produced 20 lenses, together with a further 47 candidates (where detection of a single arc indicated lensing by the primary galaxy). The approach by Jackson (2008), on the other hand, was to *manually* analyse all 285,423 galaxies, resulting in the further identification of two definite gravitational lenses, a third highly probable system, and a further 112 candidates.

The adjustments to the model as described in this section lead to a prediction of 122 detectable lenses, the key details of which are listed in Table 4.7. These results are plausible given the studies by Faure et al. (2008) and Jackson (2008), and suggest further lensing systems in the COSMOS field have yet to be confirmed.

Table 4.7: COSMOS Predictions

Parameter	Mean	Median	Variance
Lens redshift	0.77	0.70	0.16
Source redshift	2.20	2.10	0.82
Einstein radius (arcsec)	0.72	0.63	0.17
Velocity dispersion (km/s)	217	217	2517
Source magnitude	26.60	26.79	1.07
Magnification	5.44	4.35	10.9

4.3 WFIRST

In this section, we consider an application of the model, for the first time, to predict the number of strong gravitational lenses discoverable by WFIRST. For the purpose of this exercise, only filter band J_129 will be considered, as this band has the greatest depth (26.9 AB).

The survey parameters for WFIRST, as entered in the *Surveys.py* module, and their values are displayed in Table 4.8; most of this data was sourced from the website *https://wfirst.gsfc.nasa.gov*, with supplementary information provided in private correspondence with the WFIRST project team and with Tom Collett.

Table 4.8: WFIRST Parameters

Parameter	Value
Survey area	2,000 sq. deg.
Filter band	J_129
Pixel size	0.11 arcsec. per pixel
Side	200
PSF	0.12 arcsecs.
Zero exposure time	1 second
Zeropoints	23.9
Skybrightness	23.5 mags per sq. arcsec.
Exposuretime	72,800 seconds
Gains	1
Number of exposures	5
Readnoise	0 e

The parameter *stochasticobservingdata* was also amended to read:

$$self.stochasticobservingdata = [twodJ\_129]$$

$$twodJ\_129 = numpy.array([[0.12, 23.5], [0.12, 23.5]])$$

Module Amendments

In addition to the parameter changes listed above, amendments were needed to the code of four other modules to accommodate both the WFIRST survey name and the J_129 band. In particular, 'stage one' of the original code does not read in magnitude values for the J_129 band, when importing the simulated *source* catalogue. Neither does it produce J_129 band values when simulating magnitudes for the *deflectors*. The code was therefore modified to produce values for the J_129 band by extrapolating from the available i_SDSS and z_SDSS values: that is, the difference in wavelength between i_SDSS and z_SDSS is 0.1 micron, and between z_SDSS and J_129 is 0.44 microns, so extrapolation gives us the respective magnitudes m as m_J=m_z-((m_i-m_z)*4.4).

This is purely phenomenological: it is a log linear correction over a short wavelength range and is therefore independent of galaxy SEDs. The correction is also redshift-independent because it depends only on observed i-z colour. Over short wavelength ranges, log linear or power law functions can approximate galaxy SEDs, with the advantage that this is model-independent and does not depend on population synthesis modelling or model redshift distributions.

The changes made to the modules may be summarised as follows:-

FastLensSim.py (lines 297-298, 331): WFIRST survey name and J_129 band name added.

MakeResults.py (lines 25, 33-34, 100-102, 154): WFIRST survey name added and 'scaling factor' amended.

ModelAll.py (lines 11, 39-40, 110-112, 175): WFIRST survey name added, and also extrapolation for J_129 values.

PopulationFunctions.py (lines146, 236, 328, 338, 387-390, 419, 454-459): Addition of J_129 band name and extrapolation routine for J_129 value.

Predictions & Comment

The adjustments to the model as described in this section lead to a prediction of 99,849 detectable lenses, the key details of which are listed in Table 4.9 and illustrated in the histograms of Figure 4.4.

Table 4.9: WFIRST Predictions

Parameter	Mean	Median	Variance
Lens redshift	0.61	0.55	0.10
Source redshift	1.86	1.74	0.78
Einstein radius (arcsec)	0.79	0.70	0.19
Velocity dispersion (km/s)	219	217	2458
Source magnitude	24.83	25.02	2.65
Magnification	5.45	4.22	14.4

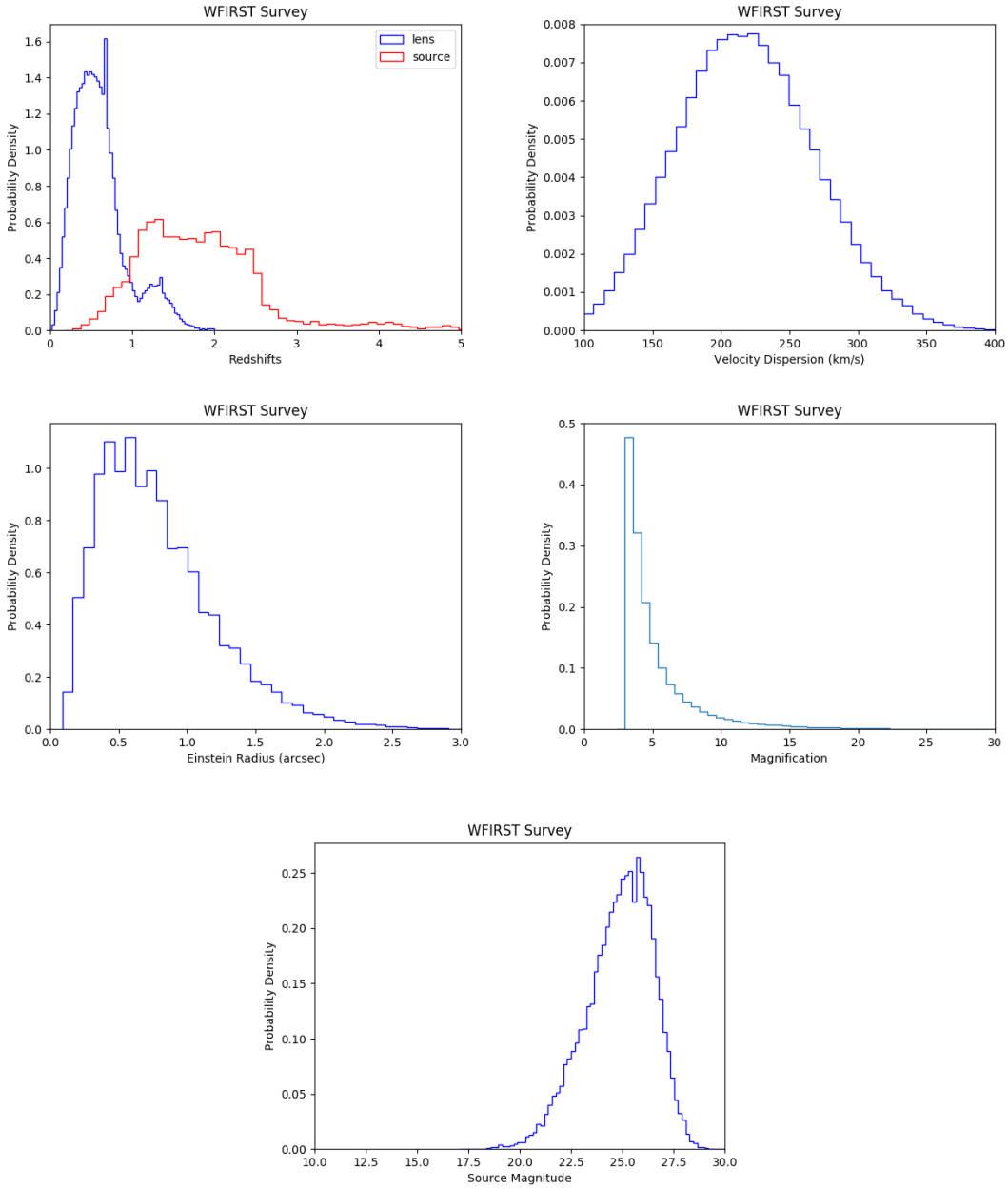

Figure 4.4: Properties of the lensing systems predicted by the model for the WFIRST survey. The source and lens redshifts are 96% and 86% of the respective values for Euclid, and the mean value of the source magnitudes is also slightly lower. However, it must be borne in mind that a number of simplifying assumptions have been made in adapting the model for WFIRST: for example, only the J_129 filter band has applied, whereas the WFIRST (HLS) survey is scheduled to carry out imaging in the Y, H and F184 bands too. Signal-to-noise considerations, also of relevance here, are discussed in the main text.

Compared to the Euclid *Wide Field* survey, for example, the predicted number of lenses suggests that the increased depth of WFIRST almost compensates for the smaller area. Significant differences between the lens properties include the lens and source redshifts, for which the respective WFIRST figures are approximately 86% and 96% of those for Euclid. At 24.8 AB, the mean value of the source magnitude is also lower in the case of WFIRST. It should be noted further that when compared to the Euclid *Deep Field* survey, WFIRST is both wider and deeper with a prediction of some 25 times more lenses.

An important consideration here concerns signal-to-noise effects. Since the WFIRST survey is intended to cover 2000 sq. deg. over a five year period, each sky position is expected to have several tens of thousands of total integration time depending on, for example, the particular survey strategy or the time spent calibrating. A nominal exposure time of approximately 73,000 seconds has therefore been chosen for this exercise, having verified that for this value a sky brightness of 23.5 mags per sq. arcsec. is consistent with a 5 sigma point source sensitivity of 26.9 AB in the J_129 band (as quoted in *https://wfirst.gsfc.nasa.gov*). This test was effected by placing a sample source flux into a 2 x 2 pixel array to simulate a point source; in reality, the zodiacal background will vary during the mission, so this analysis can only be considered indicative until the survey strategy has been determined. This does not mean however that sources as faint as 26.9 AB will be detected as lensed sources, as they will likely be extended, rather than point, sources. This represents more of a constraint on WFIRST than on Euclid, since the telescope diameter in the case of the former is larger than that of the latter, so that for any given source size the flux will be spread over more pixels. It is worth noting here that the code used in Collett (2015) calculates values for signal-to-noise ('S/N') by effectively creating an aperture around each source, such that increasing the aperture by one pixel in any direction will *decrease* the S/N of the total within the aperture: an acceptance threshold of S/N >20 is then applied within the code. As a result, the lens and source magnitude distributions for the WFIRST predictions are significantly above the nominal 5 sigma point source limit, although the point source magnitude limit has been verified in simulated images.

Finally, it should also be borne in mind that for this simplified exercise, only the J_129 band has been incorporated into the model, whereas the WFIRST (HLS) survey is planned to carry out imaging also in the Y, H and F184 bands. Furthermore, the data provided by the WFIRST team in respect of some of the parameters, e.g. skybrightness, are estimates only and are subject to change.

4.4 Discussion - Model Limitations

It should be noted at this stage that there are limitations to the model, which could prove of significance. By way of examples, we have already alluded to the exclusion of cluster lenses, whereby clusters of galaxies could give rise to a lensing effect; at present, the model has been programmed only to recognise single galaxy lenses. Clusters not only magnify sources in their integrated brightness, but they also enlarge the angular size of a distant source. A combination of adaptive optics and gravitational lensing can therefore lead to spectacular opportunities; for example, a galaxy at z=3 is typically 0.2-0.3 arcsecs across, but a magnification of 30 times enables spectroscopic data to be obtained across its enlarged image and the identification of a rotating disc (Ellis 2010). The model also excludes the possibility of 'jackpot' lenses (where a source is lensed by more than one lensing object), although 'vestigial elements' in the modules suggest coding for the latter may initially have been considered but not completed. By a similar token, secondary halos along the line of sight are neglected which, according to a recent study by Li, N. et al. (2018), may significantly affect the detectability of cluster-scale strong lensing. Other shortcomings include the assumption of elliptical galaxies only, so that other morphologies (such as spiral galaxies) are ignored, and the exclusion of lensed quasars in the source population of the model.

From the analyses conducted in this chapter, it is worth highlighting several points. Firstly, the Wide Field survey of Euclid is predicted to detect some 180,000 lenses. Although various modifications were required to correct for discrepancies in the code, both the number and the properties of the predicted lensing systems are consistent with the findings of Collett (2015). In particular, we note that the survey seems well suited to high magnification events, with the

prediction of about 33,000 of these rare systems discoverable by Euclid likely to be an underestimate due to the limitations of the model itself. The conservative nature of this prediction is also suggested by the apparent 'cut-off' of the source galaxy magnitudes at $\sim$ 28 AB, since this corresponds to the limit of the LSST catalogue. We could conclude from this that Euclid is in fact likely to identify lenses *in excess* of those predicted by Collett (2015), with the higher magnification lensing systems making worthwhile follow-up targets for deeper surveys, such as the Extremely Large Telescope ('ELT') planned for completion in 2025; resolving large numbers of source galaxies at very high angular resolutions would provide an excellent opportunity to investigate star formation near the peak of cosmic star formation history.

As far as the COSMOS predictions are concerned, although these are plausible given studies conducted elsewhere, the implication is that further lenses remain to be confirmed. In this respect, it should be borne in mind that in the absence of any detailed analysis, discrepancies could simply be the consequence of 'over-optimistic' simulations or an inability (human or machine) to accurately identify lensing events.

Finally, the above results suggest a *prima facie* conclusion that the Euclid Wide Field survey is more suitable for the detection of strong gravitational lensing than WFIRST, in that Euclid is predicted to detect a greater number of lenses. Factors behind this include signal-to-noise effects, as well as differences in survey area, the filter bands, and a number of simplified assumptions and estimates necessarily applied to the WFIRST survey parameters as part of this exercise.

Chapter 5

Cosmological Constraints

Abstract

In this chapter, I investigate the degree to which the model can be used to constrain fundamental cosmological parameters. To this end, the consequences of variations in the parameter values of the dark energy equation of state are examined, as are those for a range of values involving the matter density parameter Ω_m. The predictions of the model when run under these variations are tested for significant differences against the results produced under the standard (or Concordance) cosmology. In order to carry out this exercise in an efficient manner, as a first step the coding of the model is modified to incorporate the **astropy** package available for use with Python code[1]. In its original form, the model does not make use of **astropy** but instead relies on Collett's own module for calculating distances. Once incorporated, **astropy** significantly simplifies the running of the model under different cosmologies. The final part of this chapter examines some of the astrophysical assumptions made within the model, in particular those relating to the galaxy luminosity function, and considers whether the model's sensitivity to these could prejudice the reliability of any constraints that might otherwise be inferred. The results suggest the model is sensitive to the density parameter, and furthermore that the astrophysics assumptions in question do not, as we might have thought, 'get in the way' of this interpretation.

[1] Astropy (*http://www.astropy.org*) is a community-developed core Python package for Astronomy (Astropy Collaboration et al. 2013, Price-Whelan et al. 2018).

5.1 Incorporating Astropy

The distance calculations used throughout Collett (2015) are mainly called from the *distances.py* module, which reflects Collett's own code for specific cosmological distance formulae; exceptions are where they are created or recreated in isolation within, say, a function or attribute definition.

This section deals with the modification of this module in order to make use of the Python **astropy** package, which incorporates a wide selection of distance and related formulae for a range of different cosmologies. Rather than relying on the somewhat restrictive code within the original *distances.py* module, this is a simpler and more efficient way to generalise the model to the range of cosmologies that will be considered in this chapter.

5.1.1 Overview

Details of the steps taken to implement the **astropy** package, applied initially in the context of a flat ΛCDM cosmology, will be given in subsequent sections, but by way of overview the procedure may be outlined as follows:-

- Create the class *FlatLambdaCDM*, which is the class name required by the **astropy** package to enable the distance (and related) attributes consistent with a flat ΛCDM cosmology to be called.
- Using the existing *distances.py* attribute names (e.g. *Da, Dm, volume*), define new functions to be executed by each of those attributes such that the standard **astropy** attributes are called instead; that is, the standard **astropy** routines will return the results required rather than the existing routines within the *distances.py* module.
- Where necessary, adjust the code in any other modules where distance formulae or cosmological parameters are defined or used independently of the *distances.py* module. (There is in fact a second *distances.py* module, located in the *StellarPop* folder, but this is not used anywhere within the code so need not concern us here).

5.1.2 Amending the distances.py Module

Under the original code, this module creates a class object called *Distance*. In order to use the **astropy** package this effectively needs to be replaced by a class object called *FlatLambdaCDM*, which is recognised by **astropy** and possesses all the relevant distance attributes. However, the class object *Distance* is called in multiple areas of the original code. It was therefore considered far more efficient simply to change the properties of the *Distance* object within this module *and leave its name unchanged*, rather than to replace each instance of it throughout the code with the *FlatLambdaCDM* class object.

In order to achieve the necessary change of properties, the *Distance* class object must therefore be re-defined as a sub-class of *FlatLambdaCDM*. This in turn requires an initialisation command within the initialisation of *Distance* that specifies the values of cosmological parameters to be applied to the *FlatLambdaCDM* head class. These values are specified in, and read from, a parameter called *cosmo* which by default is given as the following array:

$$cosmo = [0.3, 0.7, 0.7]$$

where $cosmo[0], cosmo[1], cosmo[2]$ correspond to the cosmological parameters Ω_m, Ω_Λ and h respectively. It should be noted that there is a typographical error in Collett (2015) which refers to an assumed value of $\Omega_m = 0.7$; this should read $\Omega_m = 0.3$. We define $H_0 \equiv h \times 100\,\mathrm{km\,s^{-1}\,Mpc^{-1}}$.

For an object of class *FlatLambdaCDM*, the original functions of the *distances.py* module are listed, together with the **astropy** equivalents used to replace them, in Appendix Table D.1.

Finally, we note here that **astropy** functions return 'Quantities', which comprise values *and* units; to extract the values only, each attribute has to be suffixed by the *.value* method.

5.1.3 Amendments to Other Modules

MakeLensPop.py

(a) **lines 149/60**

The cosmological parameters are restated as a variable *cosmo* and this is passed as an argument of the *LensSample* class object; the values of the parameters in this variable overwrite the default values in the initialisation of the *LensSample* class.

(b) **line 66**

The values of the cosmological parameters in the variable *cosmo* are carried through to create a *Distance* class object; these values overwrite the default values in the initialisation of the *Distance* class.

Both the above therefore require amending, namely: (i) to remove the *cosmo* variable as an argument in the creation and initialisation of the *LensSample* class object, and (ii) to remove it as the argument of the *Distance* class object (it is not a valid argument in **astropy**).

ModelAll.py

(a) **line 10**

As above, the cosmological parameters are restated as a variable *cosmo* and this is passed as an argument of the *LensSample* class object.

An amendment is required here to remove the variable *cosmo* as an argument in the creation of the *LensSample* class object.

PopulationFunctions.py

(a) **lines 9-12**

The cosmological parameters are restated as a variable *cosmo* and this is passed as an argument of the *RedshiftDependentrelation* class object, and subsequently as an argument of the function *beginRedshiftDependentRelation*.

(b) **line 146**

The cosmological parameters are restated as a variable *cosmo* and this is passed as an

argument of the *LensPopulation* class object and subsequently as an argument of *beginRedshiftDependentRelation*.

(c) **line 336**

The cosmological parameters are restated as a variable *cosmo* and this is passed as an argument of the *SourcePopulation* class object and subsequently as an argument of *beginRedshiftDependentRelation*.

(d) **line 491**

The cosmological parameters are restated as a variable *cosmo* and this is passed as an argument of the *AnalyticSourcePopulation*class object (although this does not seem to feature elsewhere in the code) and subsequently as an argument of *beginRedshiftDependentRelation*.

Amendments are therefore required here to remove the references to the *cosmo* variable. This means removing it as an argument of the above class objects, as well as removing it as an argument from the function *beginRedshiftDependentRelation* (which is called by those classes).

5.1.4 Using Astropy

One of the major benefits of using the **astropy** package, as opposed to the original *distances.py* module alone, is that any changes that require to be made to the cosmological parameters (e.g. for testing different cosmological models) can be effected by directly amending the arguments of the single *cosmo* variable given in the *distances.py* module. There is no need to search for, or amend, any other instances of the parameters in the code.

It is also helpful that in addition to flat CDM cosmology, **astropy** can readily accommodate other cosmologies by using a class other than *FlatLambdaCDM*. The class *LambdaCDM*, which allows for independent values for both Ω_m and Ω_{de} (and hence a non-flat cosmology), and the class *wCDM*, which allows not just for independent values of Ω_m and Ω_{de} but also for w (and hence variations to the dark energy equation of state, as discussed below), are two examples of classes that may be used instead of the *FlatLambdaCDM* class to which we referred above. Apart from the need to specify the additional key parameters, no other changes to the code are required.

An example of a *distances.py* module that has been amended to make use of the **astropy** package, and which has been used in applications of the model in this project, is given in Appendix D.

It should be noted that once the amendments were made to the *distances.py* module, the model was run again on the Wide Field and Deep Field surveys. The results were then compared with those obtained for the two surveys prior to the introduction of the **astropy** package. The respective results of both sets (ie. pre- and post- **astropy**) are shown in Appendix D.2 and, as anticipated, may be confirmed by inspection to be consistent.

5.2 The Cosmological Parameters

In this section, we investigate the extent to which the model may be used to constrain the values of key cosmological parameters. We consider firstly the parameters making up the dark energy equation of state $p = w\rho$, before turning to the cosmological density parameter Ω.

5.2.1 Dark Energy Equation of State (EoS)

As discussed previously, the original code assumed a standard (ΛCDM) cosmology, which was effectively 'built in' to the *distances.py* module. However, modifying that module to incorporate the functionality of the **astropy** package has made it possible to readily run the code under a number of alternative cosmologies, including those which might enable us to impose constraints on the dark energy equation of state.

In addition to the ΛCDM cosmology, we therefore consider here the equation of state (EoS) for two alternative cosmologies, known as *wCDM* and *CPL* (or $w_0 w_a CDM$) cosmologies. The equation of state for each of these three cosmologies may be summarised and compared as follows:-

- **ΛCDM**

$$p_\Lambda = -\rho_\Lambda$$

where Λ represents the Cosmological Constant, and p and ρ are the pressure and density respectively.

- **wCDM**

$$p_{de} = w\rho_{de}$$

where w is a constant, sometimes referred to as the *EoS parameter of dark energy*; note that Λ**CDM** is a special case where $w = -1$, with dark energy synonymous with Λ.

- **CPL**[2] **(or** $\mathbf{w_0 w_a}$**CDM)**

$$p_{de} = w(z)\rho_{de}$$

where

$$w(z) = w_0 + \frac{w_a z}{(1+z)}$$

and w_0, w_a are free parameters; note the dependence of w on redshift z.

With the above in mind, the (modified) model was run under the $wCDM$ and CPL cosmologies using both the best fit and 2-sigma variations in the Planck DES parameter values provided by Xu & Zhang (2016); for the 2-sigma variations, the values input to the model correspond to four extrema identified from a visual inspection of the 2-sigma (Ω_m, w) and (w_o, w_a) contours of the $wCDM$ and CPL cosmologies respectively. The results of these runs could then be compared to those obtained under the standard (or Concordance) cosmology. For the purpose of this analysis, it should be emphasised that we are considering only *flat* cosmologies (ie. where $\Omega = \Omega_m + \Omega_{de} = 1$); these correspond to the **astropy** class objects FlatwCDM and Flatw0waCDM.

For the sake of consistency when comparing these three cosmologies, *best-fit Planck* data for the flat ΛCDM model has been used in place of the approximated values ($\Omega_m = 0.3$ and $h = 0.7$)

[2] *'Chevallier-Linder-Polarski'.*

for the standard cosmology assumed in the original code by Collett (2015).

Parameter Values

Listed in Table 5.1 are the parameter values for each of the cosmologies under which the model has been run; the *limits* referred to are the four 2-sigma extrema in the Planck DES parameter contours mentioned above.

Table 5.1: Tested Cosmologies (Planck)
source: Xu & Zhang (2016)
(note: $\Omega = \Omega_m + \Omega_{de} = 1$)

Cosmology	Fit	Parameters
flat ΛCDM	best-fit	$\Omega_m = 0.324, \quad h = 0.667$
flat CPL	best-fit	$\Omega_m = 0.326, \quad w_0 = -0.969, \quad w_a = 0.007, \quad h = 0.663$
flat CPL	limit 1	$\Omega_m = 0.326, \quad w_0 = -1.14, \quad w_a = 0.60, \quad h = 0.663$
flat CPL	limit 2	$\Omega_m = 0.326, \quad w_0 = -0.78, \quad w_a = -0.90, \quad h = 0.663$
flat CPL	limit 3	$\Omega_m = 0.326, \quad w_0 = -0.93, \quad w_a = 0.75, \quad h = 0.663$
flat CPL	limit 4	$\Omega_m = 0.326, \quad w_0 = -0.98, \quad w_a = 0.22, \quad h = 0.663$
flat wCDM	best-fit	$\Omega_m = 0.326, \quad w_0 = -0.964, \quad h = 0.662$
flat wCDM	limit 1	$\Omega_m = 0.312, \quad w_0 = -1.02, \quad h = 0.662$
flat wCDM	limit 2	$\Omega_m = 0.341, \quad w_0 = -0.915, \quad h = 0.662$
flat wCDM	limit 3	$\Omega_m = 0.323, \quad w_0 = -0.92, \quad h = 0.662$
flat wCDM	limit 4	$\Omega_m = 0.333, \quad w_0 = -1.01, \quad h = 0.662$

Methodology & Results

In comparing the predictions of the model under the different cosmologies, the objective was to discover the extent to which there are significant differences which would enable us to constrain the EoS parameter values. Specifically, this means comparing histograms of the predictions for the Euclid survey under each of these cosmologies (namely, running the code in its entirety for each set of parameter values, for each cosmology 'class object').

Initially, the histograms were binned on lens redshift (zl), source redshift (zs), magnification (mag), and Einstein radius (b). The comparisons were undertaken using a Chi-Square (χ^2) analysis, with testing against the Planck best-fit flat ΛCDM cosmology (or Concordance cosmology)

as the null hypothesis.

A helpful discussion of histogram comparison techniques such as that employed here may be found in Bityukov et al. (2013*a*,*b*) and Scott (2003) but, by way of brief explanation, the bins in each histogram are treated as 'observations': if the histograms are from the same distribution, each bin will typically have a χ^2 value of about 1. The central limit theorem tells us that for a large number of bins (as in the cases here), the average χ^2 value will have an approximately normal distribution, with a standard deviation of $\sigma/\sqrt{N}$, where σ is the standard deviation of the χ^2 distribution and N is the number of observations. For r degrees of freedom, the variance is given by $r * (r + 2)$, so a single χ^2 will have a mean of 1 and $\sigma = \sqrt{3}$. (See, for example, *http://mathworld.wolfram.com/Chi-SquaredDistribution.html*). Importantly, the null hypothesis that two histograms are the same can thus be tested by comparing the observed mean χ^2 with a Gaussian of mean = 1 and standard deviation $\sqrt{3}/\sqrt{N}$. Under such circumstances, a z-test may be considered appropriate for comparing the histograms. An example of the source code written to carry out this analysis is given in Appendix E.2.

It may be noted that the data distributions may alternatively be compared using the Kolmogorov-Smirnov test. However, it was felt more effective here to use binned data (and the Chi-Square analysis described above), since doing so allows any discrepancies to be more readily identified and localised; furthermore, the number of observations in this case means that binning leads to only a negligible loss of data.

In the event, this analysis carried out did *not* produce any significant differences in the cosmologies. The implication of this is that, on the face of it and as far as using the Planck DES data is concerned, binning and analysing the results *in this way* does not lead us to impose any constraints on the cosmological parameters.

However, an alternative procedure for binning the data was then applied. This involved binning on lens redshift, source redshift, and on the Einstein radius divided by the square of the velocity dispersion (b/σ^2). The rationale behind the latter bin is that it is equal - up to a constant - to

the distance ratio D_{ls}/D_s, where D_{ls} and D_s are the source-lens and observer-source angular diameter distances respectively, which constitutes an observable quantity and is well-defined for any given cosmology (e.g. Leaf & Melia 2018). The magnification bin was disregarded as it is redundant: it is not independent of the other bin parameters. By binning in this (3-D) manner, we find that the cosmologies *do* give rise to histograms that have some visually noticeable differences compared to the standard cosmology, as shown in Figures 5.1 and 5.2 , and in some cases these are of statistical significance.

Table 5.2: EoS Likelihoods (vs Planck DES Best-Fit ΛCDM)

Cosmology		Predicted Lenses	Z-test	p_values
flat ΛCDM	best fit	157,983	-	-
flat CPL	best fit	158,691	0.7376 (0.5023)	0.2304 (0.3077)
flat CPL	limit 1	161,703	6.9908 (0.7848)	0.0 (0.2163)
flat CPL	limit 2	157,791	8.2396 (0.4258)	0.0 (0.3351)
flat CPL	limit 3	146,682	38.30 (0.9193)	0.0 (0.179)
flat CPL	limit 4	156,759	3.5162 (0.1685)	0.0002 (0.4331)
flat wCDM	best fit	159,561	0.0551 (0.2446)	0.478 (0.4034)
flat wCDM	limit 1	165,609	1.8671 (0.5776)	0.0309 (0.2818)
flat wCDM	limit 2	153,192	4.5318 (0.8622)	0.0 (0.1943)
flat wCDM	limit 3	157,545	0.6515 (0.1534)	0.2574 (0.4391)
flat wCDM	limit 4	159,381	1.5074 (0.1991)	0.0658 (0.4211)

The quantitative results of the histogram comparisons are shown in Table 5.2, where the figures in parentheses refer to the initial (4-D) bins. Although these may allow us to infer some constraints on the EoS parameters, there remains a concern here that the size of the population might be a limiting factor on the reliability of these results. Whilst the population of idealised lenses is of a healthy order $\sim 10^6$, the *ModelAll.py* module (ie. 'Stage Two') takes only approximately one-tenth of this to determine the detectability by Euclid, subsequently simply multiplying the result by ten before reporting it (the issue with scaling in this manner, outlined already in sec. 3.2.13, will be addressed again later in this project). This fraction may be amended manually, but it was found that just doubling it increased the running time of the code by over 24 hours, which would have rendered running the full model for a series of different cosmologies impractical.

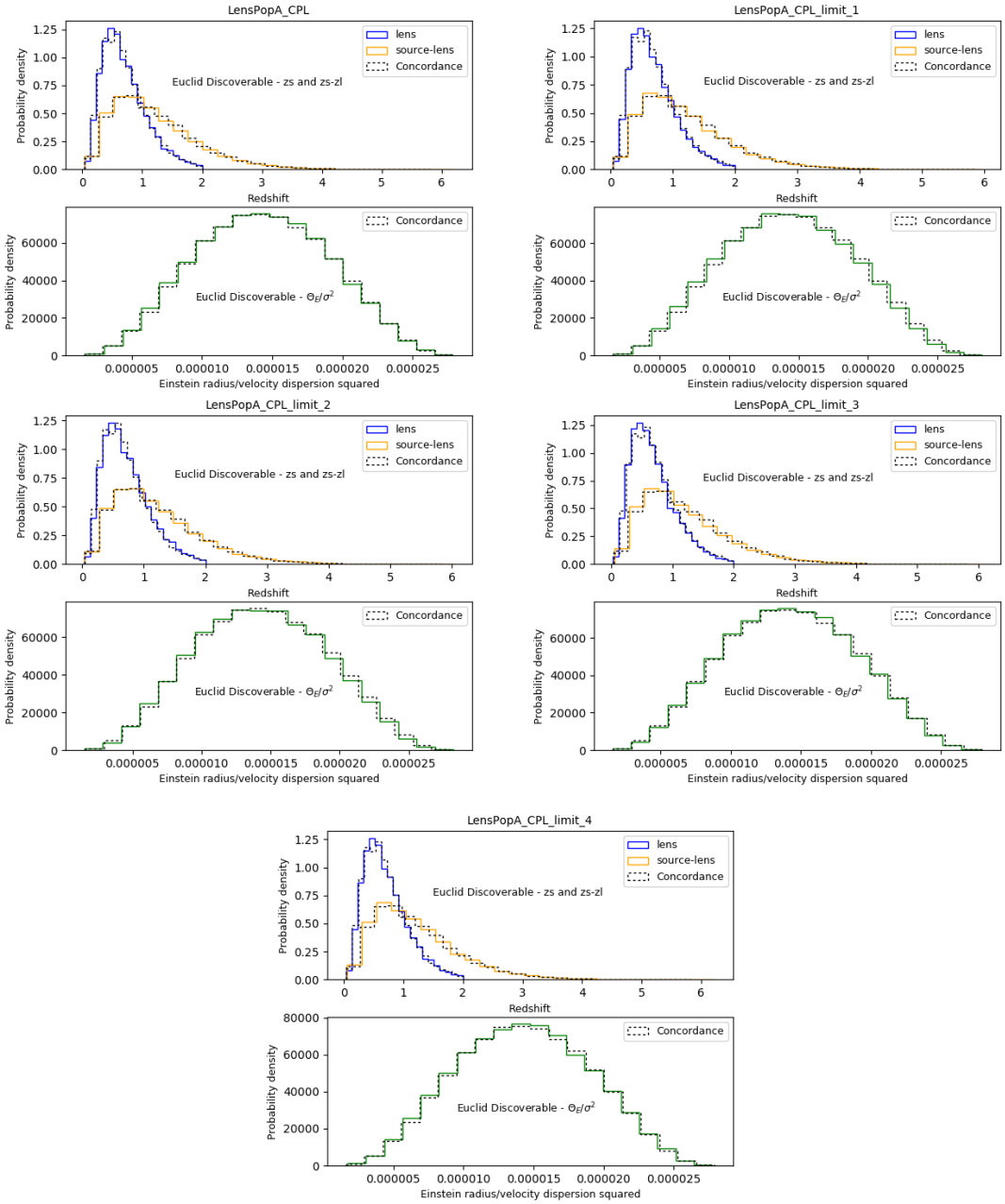

Figure 5.1: Properties of lensing systems predicted by the model under different CPL cosmologies. Results obtained under the Concordance (Planck DES Best-Fit ΛCDM) cosmology are shown for comparison and some noticeable differences are apparent.

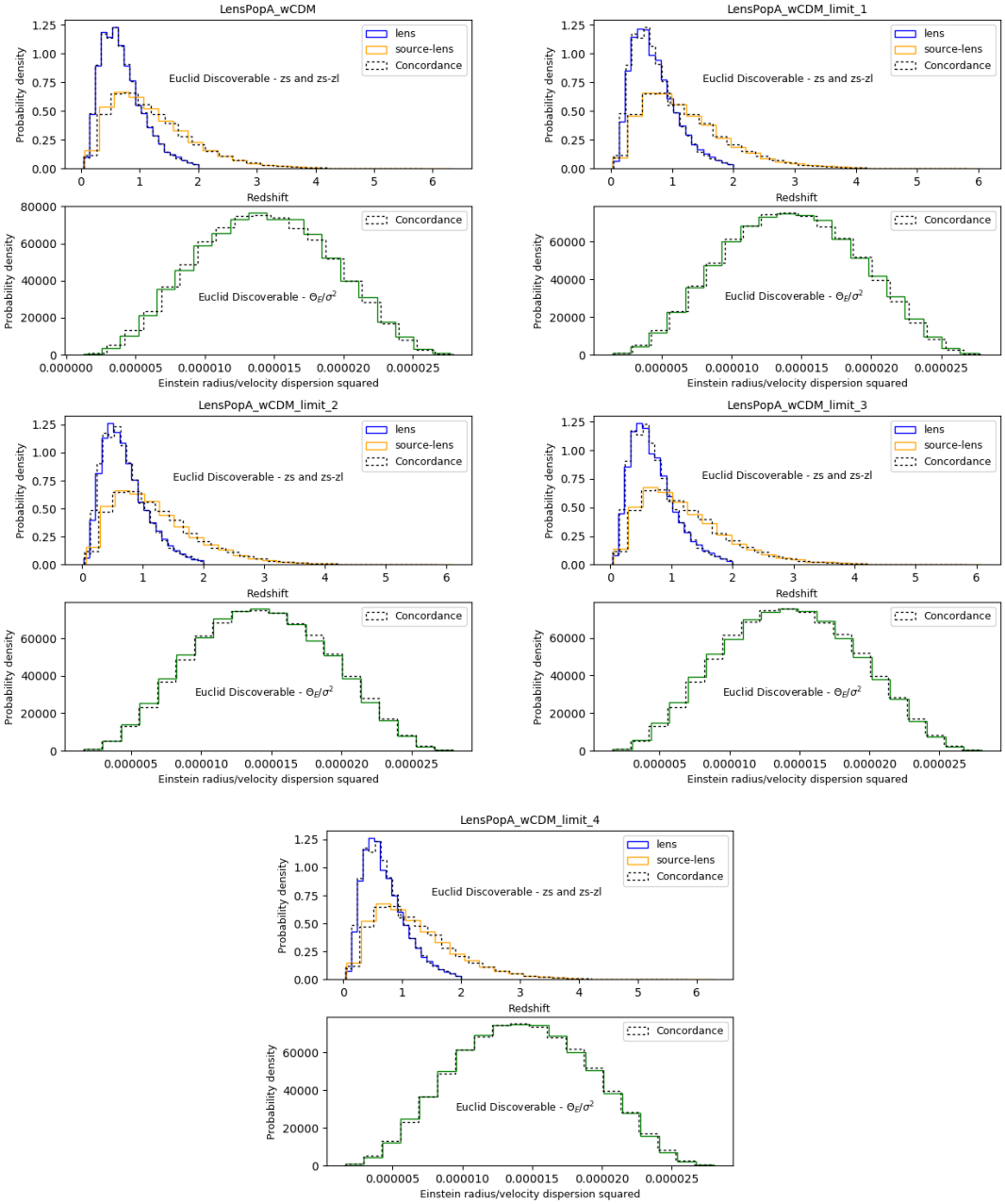

Figure 5.2: Properties of lensing systems predicted by the model under different wCDM cosmologies. As with the CPL cosmologies illustrated in Figure 5.1, results obtained under the Concordance (Planck DES Best-Fit ΛCDM) cosmology are shown here for comparison and again some noticeable differences are apparent.

It was therefore considered worthwhile measuring the extent to which the idealised lenses *alone* could be used to assess the effects of varying the cosmologies. Part of the motivation for this is that the module *ModelAll.py* does not directly reference the cosmological parameters, and instead relies entirely on the properties returned for the idealised lenses by the *MakeLensPop.py* module; in other words, for example, if there is no significant difference between sets of *idealised lenses*, then it is reasonable to expect there will be no significant difference between sets of lenses detectable by Euclid.

Consequently, further analyses were performed on the idealised lenses alone (as opposed to the detectable lenses), using not only the Planck DES data above, but also, as a separate exercise discussed below, when investigating a range of values for the density parameter Ω_m in a flat ΛCDM cosmology.

Idealised Lenses & EoS Constraints

For the EoS constraints therefore, comparisons were carried out on a sample of the idealised lenses using the same binning technique as for the Euclid detectable lenses above. The corresponding histograms are shown in Figures 5.3 and 5.4, with the quantitative results listed in Table 5.3. By inspection, we note these are broadly consistent with the results obtained from the sample of detectable lenses, although there is an exception in the case of the *flat wCDM - limit 1* cosmology. For the latter, the idealised lenses imply an 8σ difference, compared to a difference of just under 2σ recorded for the detectable lenses.

5.2.2 The Density Parameter

Following on from the above discussion on the dark energy equation of state, as a separate (albeit not unrelated) exercise, we continue now to examine whether the model can be called upon to impose any constraints on the components of the cosmological density parameter Ω.

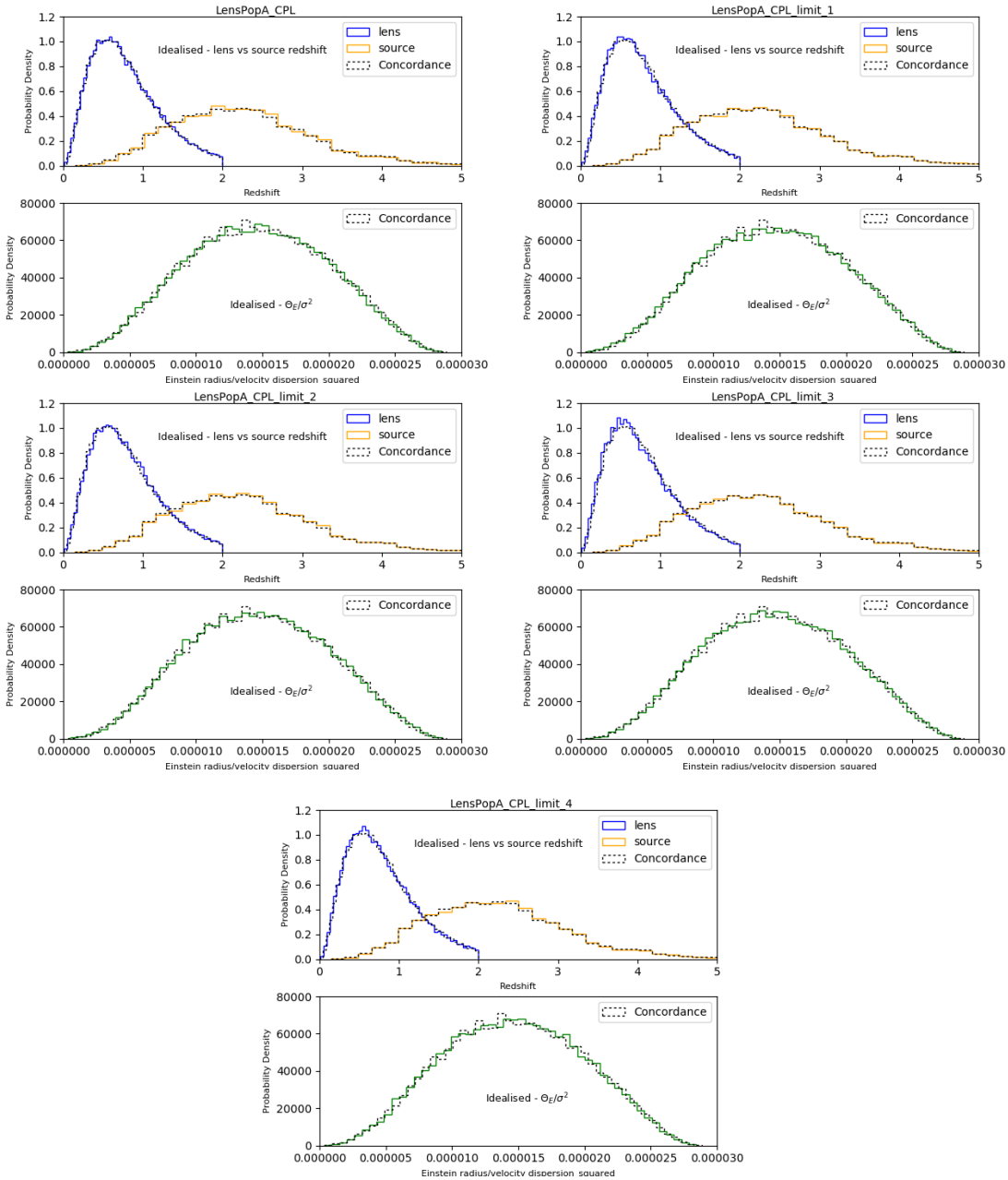

Figure 5.3: Properties of lensing systems predicted by the model under different CPL cosmologies, based on a sample of *idealised* lenses rather than *detectable* lenses. The results obtained here for the former are broadly consistent with those of the latter.

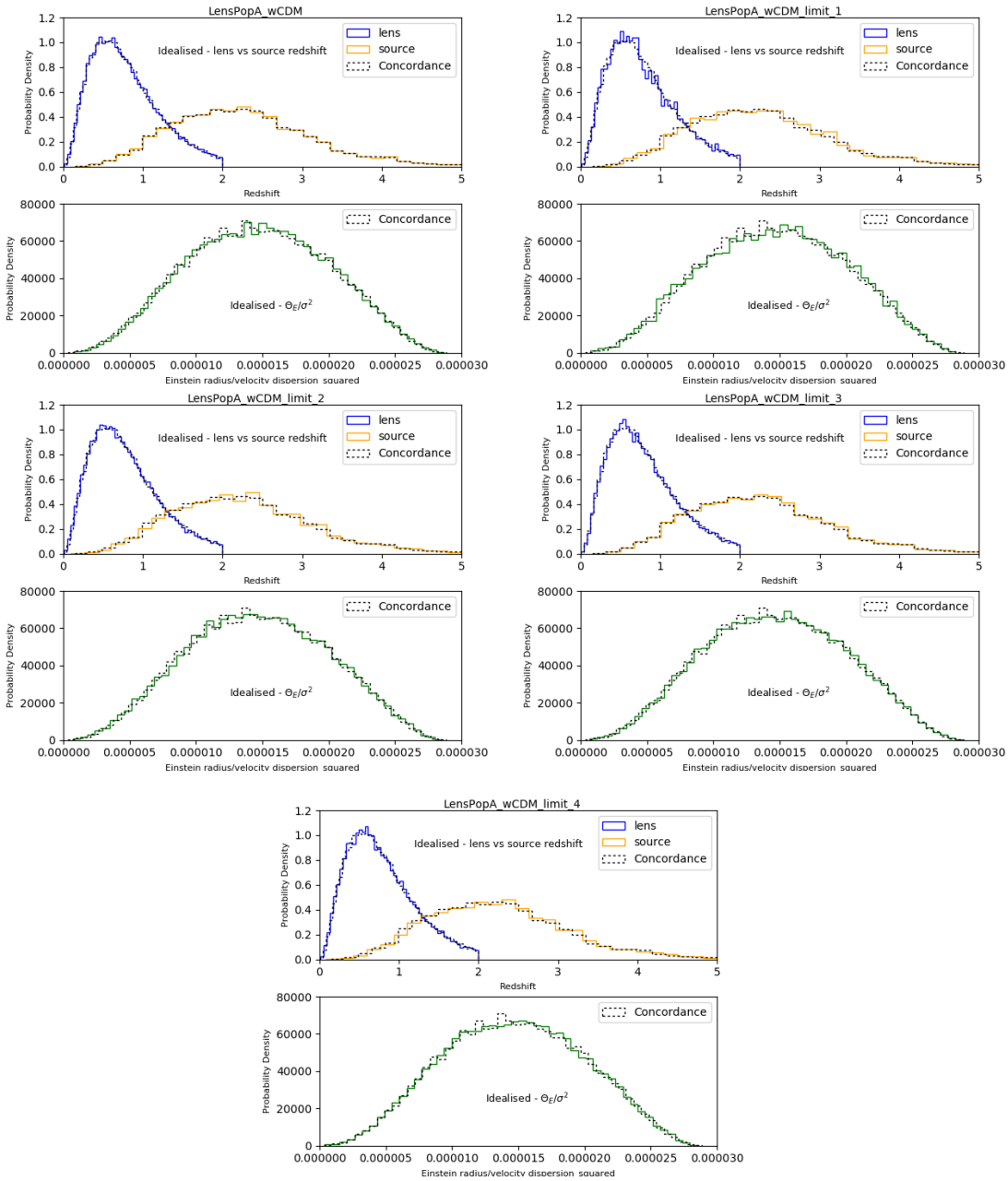

Figure 5.4: Properties of lensing systems predicted by the model under different wCDM cosmologies, based on a sample of *idealised* lenses rather than *detectable* lenses. As with Figure 5.3, with one exception (namely, the *flat wCDM - limit 1* cosmology), the results obtained for the former are again broadly consistent with those of the latter.

Table 5.3: EoS Likelihoods (vs Planck DES Best-Fit ΛCDM)
Idealised Lenses (Sample Size 50,000)

Cosmology		Z-test	p_values
flat ΛCDM	best fit	-	-
flat CPL	best fit	0.2965	0.3834
flat CPL	limit 1	8.3637	0.0
flat CPL	limit 2	9.4671	0.0
flat CPL	limit 3	38.70	0.0
flat CPL	limit 4	2.258	0.012
flat wCDM	best fit	0.0294	0.4883
flat wCDM	limit 1	8.2668	0.0
flat wCDM	limit 2	4.2157	0.0
flat wCDM	limit 3	0.1715	0.4319
flat wCDM	limit 4	1.8363	0.0332

For these purposes, we consider only flat ΛCDM cosmology with, once again, the Planck best-fit flat ΛCDM model representing the standard or Concordance cosmology. It is important to note that $\Omega = \Omega_\Lambda + \Omega_m = 1$ under the assumption of a flat ΛCDM cosmology. Much of the following analysis is conducted in terms of the parameter Ω_m; as there is only one free variable, this is functionally equivalent to examining the dependence on Ω_Λ.

Bearing in mind the issue of sample sizes, for this exercise we consider firstly populations of *idealised* lenses, before turning to the smaller, but arguably more relevant, populations of *detectable* lenses. As with the histograms discussed above, data obtained for lenses over the range $0.20 < \Omega_m < 0.95$ were binned on lens redshift, source redshift, and on the Einstein radius divided by the square of the velocity dispersion. By *quantitatively* comparing histograms for each value of Ω_m against that of the Planck best-fit (or Concordance) value, we are able to construct a likelihood curve: that is, we can plot the p_value associated with testing any value of Ω_m in the quoted range against the Planck best-fit value. The method used to fit and plot such a curve is discussed in Appendix E.3.

Ω_m Results - Idealised Lenses

Initial Data

The first approach considered was simply to run 'Stage One' of the model with values of $0.20 < \Omega_m < 0.95$, and then to carry out a histogram analysis on the resultant sets of idealised lenses. However, this method ignores the potentially significant errors in the derivation of the number of potential deflectors, resulting from the dependence of the comoving volume on extreme values of the density parameter (see section 3.2.6). Further analysis of this particular set of data was therefore not pursued, although the corresponding histogram plots are displayed in Appendix E.1 for the sake of completeness.

Volume-Adjusted Data

As described in section 3.2.6, adjusting cosmological parameters such as the density parameter had an impact on the number of potential deflectors that I had not anticipated. On inspection, it became clear that this is because the number density function assumed within the model is a function of the comoving volume, which depends on those parameters; and, although the number of deflectors therefore changes within the code according to the cosmology, the model actually relies on that number having a pre-determined *fixed* value. In other words, the number of deflectors in the code should not in fact be dependent on the cosmology.

In this approach, we correct for the number of deflectors to ensure it remains constant, regardless of the value of Ω_m, before proceeding with a histogram analysis of the idealised lenses over the range $0.20 < \Omega_m < 0.95$. The modifications required to correct the code are detailed in Appendix B.

A sample of the resultant histogram plots are shown in Figures 5.5 and 5.6. Visually, there is very little difference between the histograms of each cosmology compared to that of the Concordance cosmology. This can be explained by noting that if the number of deflectors is the same for each cosmology, then adjusting the value of the density parameter in the code will only influence the results to the extent that it affects the Einstein radius (which depends on angular diameter dis-

tances). To identify an idealised lens, the code requires a source galaxy to lie within the Einstein radius, but the latter tends to be constant where the source redshift is much greater than the lens redshift, and decreasing as the redshift values approach one another (Serjeant 2012). The upshot of this is that the histograms reflect only a small difference in the redshift distribution of the deflectors for each cosmology.

With regard to the likelihood curves, those corresponding to sample sizes ranging from 1,000 to 200,000 are displayed in figure 5.7; as we would anticipate, their distributions tend as a function of size to centre more sharply in the neighbourhood of the Planck best-fit value for Ω_m.

Redshift-Filtered Data

We noted above that variations in the value of the density parameter, within the model's code, affects the lensing statistics principally via their impact on the Einstein radius, and that this impact is only of significance when the source and lens redshifts approach one another. Based on this, we may therefore choose to exclude lenses where zs $\gg$ zl, and instead examine the impact of varying that parameter only on systems where zs - zl < 1, since any discernible effect is generally restricted to these cases. A plot of the data filtered in this way is shown in Figure 5.8, and by inspection is consistent with that premise.

Ω_m Results - Detectable Lenses

The option to filter the lensing systems as above has a major practical benefit as far as executing the model's code is concerned. Up to this point, and for the reasons discussed earlier, the likelihood curves have been based on idealised lenses only - rather than those predicted as *detectable* by Euclid. It will be recalled that the methodology means only 10% of the (full sky) idealised lenses are tested against the criteria for Euclid detectability; a weighting is then applied within the code, which effectively means the sample that satisfies those criteria is simply multiplied by a factor of 10 - resulting in a prediction of lenses that Euclid would detect *were it able to survey the whole sky* - before ultimately being scaled down again to reflect the fraction of the sky that Euclid will actually see. This is clearly not ideal from a statistical sampling point of view, but

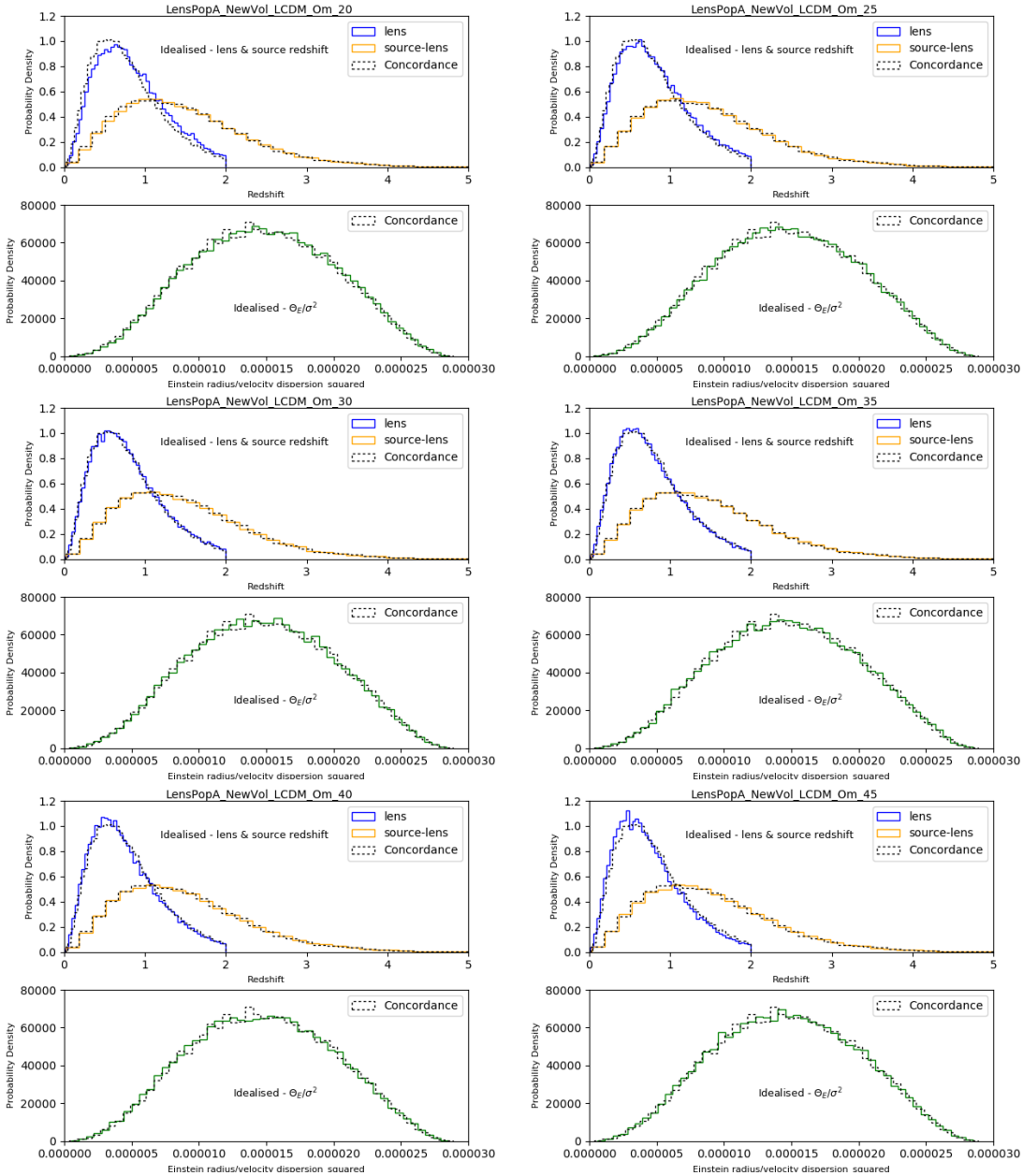

Figure 5.5: Properties of lensing systems predicted under a range of different values for Ω_m in a flat ΛCDM cosmology (ie. where $\Omega_m + \Omega_\Lambda = 1$). The results follow from modifications to the model as discussed under *Volume-Adjusted Data* in section 5.2.2. The lack of any significant difference between the histograms of each cosmology compared to that of the Concordance cosmology is a consequence of only a small difference in the redshift distributions of the deflectors for each. (See also Figure 5.6).

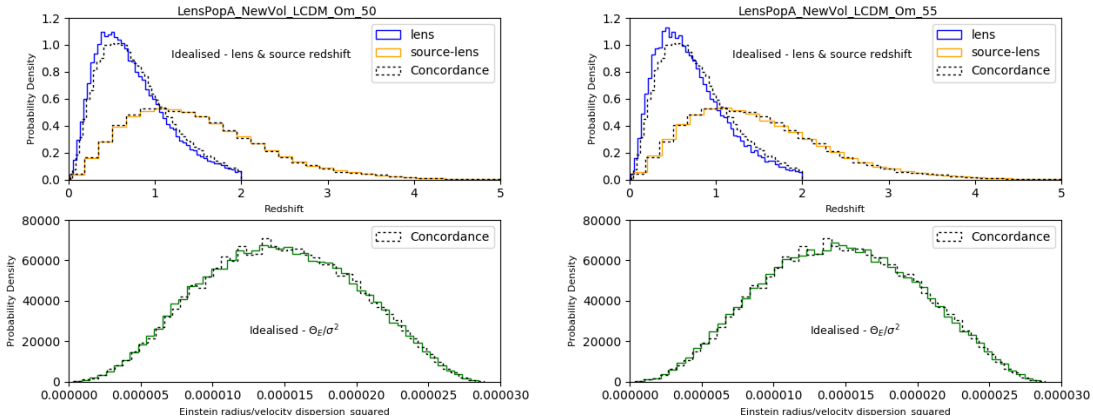

Figure 5.6: Continued from Figure 5.5: properties of lensing systems predicted under different values for Ω_m in a flat ΛCDM cosmology.

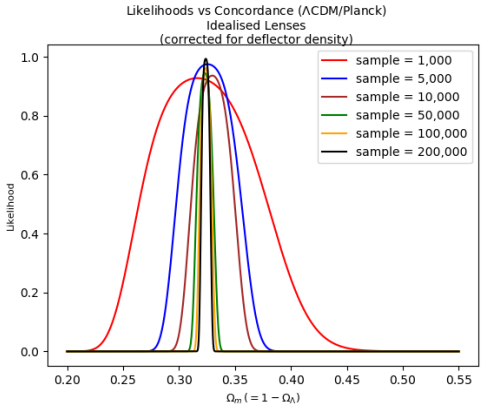

Figure 5.7: Idealised Lenses (Volume-Adjusted) & Likelihood Plot for Ω_m. Increasing the sample size leads to a narrowing of the distribution for the density parameter in the neighbourhood of the Planck best-fit value, as anticipated.

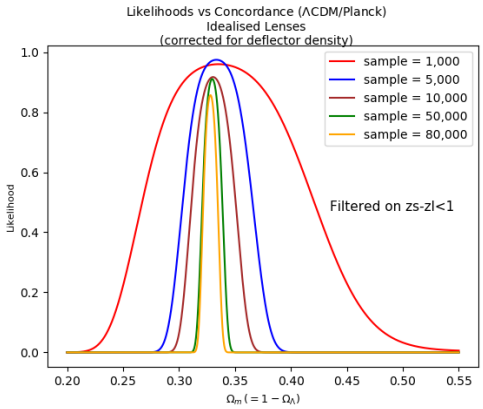

Figure 5.8: Idealised Lenses (Redshift-filtered) & Likelihood Plot for Ω_m. Only lensing systems where zs - zl < 1 have been included in this plot: the appearance of the curves is consistent with the premise that any discernible effect of changes to the density parameter is mainly restricted to such cases.

is understandable from a run time perspective, since attempting to test the full sky number of idealised lenses (about 11 million) against the detectability criteria (and then reducing it by the fraction of surveyed sky) would require an impractical amount of computer time; a single typical run takes approximately 20 hours.

By filtering the data on redshift, with no significant loss of information (at least as far as constraining the density parameter is concerned), the run time of the model can be reduced by roughly one third. It was therefore considered reasonable, and sensible, to modify the code in 'Stage Two' to proceed by filtering the *full sky* of idealised lenses on redshift, before scaling down the sample to reflect the survey area, and then applying the Euclid detection criteria. The run time is largely unaffected by this 're-ordering', but importantly this procedure has the benefit of predicting detectable lenses based on sampling from the original full sky data (rather than only 10% of it), which is more akin to the actual physical performance of Euclid.

Amendments to the code in this way involve only the *ModelAll.py* and *MakeResults.py* modules, and these modifications are detailed in Appendix E.4. Once implemented, the model was run and the resultant set of likelihood curves - namely, those based on the detectable (rather than idealised) lenses - is illustrated in Figure 5.9. By inspection, this plot suggests a constraint on

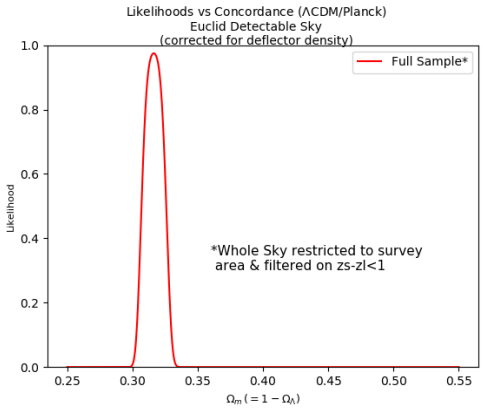

Figure 5.9: Euclid Detectable Lenses & Likelihood Plot for Ω_m. This plot is based on a modfication to the code to filter the *whole sky* of idealised lenses on redshift (such that zs - zl < 1), before scaling down to reflect the survey area and then applying the Euclid detection criteria. 'Reordering' the process followed by the model in this way leads to an arguably more realistic set of results, from which we can infer a plausible constraint on the density parameter of approximately ± 0.02.

Ω_m of approximately ± 0.02; that is,

$$0.30 < \Omega_m < 0.34$$

5.3 Astrophysics Assumptions

So far, examining the potential for constraining cosmological parameters has essentially meant 'tweaking' those parameters in the model, and testing whether the resultant predictions differ significantly from one another. It is on that basis that we have been able to infer a constraint on the density parameter, as discussed above.

This approach however is necessarily simplistic, and although an in-depth analysis is restricted by the scope of this project, it is important nevertheless to highlight the sensitivity of the model to the astrophysical assumptions that underlie it. The motivation for this is that any differences in the model's predictions may be overwhelmingly due to errors in the assumptions regarding, say, the galaxy evolution or luminosity functions rather than differences brought about by vari-

ations in the values of the cosmological parameters.

The next step therefore is to investigate the sensitivity of the model to those assumptions, and in doing so establish the extent to which 'astrophysics gets in the way'[3] of using the model to constrain the cosmologies.

We start by considering the form of the density function, from which the potential lens population is determined (ie. 'Stage One'). Collett's code makes use of the functional relationship described in Choi et al. (2007), stating:

$$dn = \phi_*(\frac{\sigma}{\sigma_*})^\alpha exp[-(\frac{\sigma}{\sigma_*})^\beta]\frac{\beta}{\Gamma(\alpha/\beta)}\frac{d\sigma}{\sigma}$$

where $\phi_* = 8.0 \times 10^{-3}h^3Mpc^{-3}$, $\sigma_* = 161kms^{-1}$, $\alpha = 2.32$ and $\beta = 2.67$.

The sensitivity of the code to this relationship can be tested therefore by reference to the upper and lower limits, quoted for the latter three parameters in the same study (Choi et al. 2007); namely,

$$\sigma_* = 161 \pm 5kms^{-1} \qquad \alpha = 2.32 \pm 0.10 \qquad \beta = 2.67 \pm 0.07.$$

To test the model's sensitivity, each of the three parameters was adjusted one at a time to its extreme values, whilst keeping the other two parameters unchanged. Within the code, this required amendments only to the definitions of the function *Phi* in the *MakeLensPop.py* and *PopulationFunctions.py* modules. The results of the (six) runs of the model were binned as before on source redshift, lens redshift, and Einstein radius divided by the square of the velocity dispersion. Again as before, these results were tested against the Concordance cosmology - representing the null hypothesis - by means of a Chi-Square analysis: in short, a rejection of the null hypothesis is an indication that the model's sensitivity to astrophysics (and to the density function in particular) prejudices the reliability of any constraint we might otherwise impose on the cosmological parameters.

[3]With thanks to my supervisor Stephen Serjeant for providing this effective soundbyte!

Another astrophysical assumption concerns the form of the effective radius (r_{eff}), as assumed for the light profile of the source galaxies in the model. The formulation is based on that by Mosleh et al. (2012), and is subject to a separate discussion in Appendix C. The key feature is that sizes evolve as $(1+z)^{\beta}$. In the model, the value of β is taken to be -1.2 but in the study by Mosleh et al. (2012) the value is quoted as $\beta = -1.20 \pm 0.11$. We therefore test the sensitivity of the model, as before, by running the code with β adjusted to its extreme values.

The test results of the (eight) runs for the model, with the variations to the parameters discussed (and with the redshift filter described in the previous section applied in each case), are presented in table 5.4; the number of potential deflectors and idealised lenses are also shown in the table, and serve to illustrate the immediate impact of each case. The results suggest there is *no* significant sensitivity to those variations. However, although testing by varying one parameter at a time may be helpful as a first step (it could alone have led to rejection of the null hypothesis), the next step was to examine the consequences of setting *all four* of the parameters, at the same time, to the values corresponding to their respective minimum likelihoods. The parameter values were therefore set as:

$$dn : \sigma_* = 156, \qquad \alpha = 2.42, \qquad \beta = 2.60$$

$$r_{eff} : \beta = -1.09$$

and the results are shown in table 5.5. Again however, there appears to be no significant sensitivity, with the resulting lens properties falling within 2.5σ of their Concordance values.

5.4 Discussion - Model Sensitivities

In this chapter, we have looked at the extent to which the model can be used to impose constraints on cosmological parameters. Consideration was given firstly to the dark energy equation of state, and in particular to the ω and Ω parameters as they appear in the Concordance, wCDM

Table 5.4: Likelihoods (vs Planck DES Best-Fit ΛCDM)
Parameter Variations (Individual)

Function	Parameter	Deflectors	Idealised Lenses	Z-test	p_values
as per original model		1,038,733,782	11,288,033	-	-
dn	$\sigma_* = 166$	1,065,905,277	12,777,,241	0.5787	0.2814
dn	$\sigma_* = 156$	1,009,184,661	9,918,675	1.1525	0.1246
dn	$\alpha = 2.42$	1,067,047,914	11,921,694	0.5458	0.2926
dn	$\alpha = 2.22$	1,008,984,162	10,663,994	0.4119	0.3402
dn	$\beta = 2.74$	1,032,697,107	10,664,000	0.1636	0.435
dn	$\beta = 2.60$	1,045,170,874	11,992,137	0.6095	0.2711
r_{eff}	$\beta = -1.31$	1,038,733,782	11,286,582	0.6253	0.2659
r_{eff}	$\beta = -1.09$	1,038,733,782	11,282,641	1.2283	0.1097

Table 5.5: Likelihoods (vs Planck DES Best-Fit ΛCDM)
Parameter Variation (Concurrent)

Deflectors	Idealised Lenses	Z-test	p_values
1,045,039,022	11,146,647	2.4467	0.0072

and CPL cosmologies. Based on Planck DES data, and the 2σ (Ω_m, ω) and (ω_0, ω_a) contours in the wCDM and CPL cosmologies respectively, the results suggest a prima facie case to claim that the model does constrain these parameters; this conclusion was reached initially by examining the results obtained for lenses deemed detectable by Euclid, but by way of a 'gross check', these were also found broadly consistent with the properties of the idealised lenses produced by 'Stage One' of the model.

The second area under consideration related to the density parameter Ω. Since we have been concerned only with *flat* cosmologies, for which $\Omega_m + \Omega_\Lambda = 1$, we note that the implications for Ω_m are related directly to those for Ω_Λ. The methodology initially applied here was to run 'Stage One' of the model to produce a population of idealised lenses for a range of values of Ω_m (or, equivalently, $1-\Omega_\Lambda$), and then compare the properties of those lenses against those resulting under the Concordance cosmology. Certain shortcomings of this procedure became clear during the course of the analysis, and consequently the method was adapted to produce a more reliable (and arguably more realistic) set of data. That analysis resulted in a constraint on Ω_m of ± 0.02.

Having established the sensitivity displayed by the model to the choice of cosmology, we then examined the extent to which this could be diluted by the sensitivity of the model to the *astrophysical assumptions* within it. In other words, significant differences in the properties of the lensing systems predicted by the model could be the result of errors or variations in the accuracy of, say, the luminosity or density functions built into the model, rather than variations in the values of the cosmological parameters. To investigate this, the parameters for both the density function (Choi et al. 2007) and the effective galaxy size (Mosleh et al. 2012) were varied according to their limits, and the properties of the resulting lens predictions tested against the Concordance cosmology. The conclusion from this approach was that the model is *not* particularly sensitive to the astrophysical assumptions.

The scope of this project has necessarily restricted the depth of analysis. In the case of the dark energy equation of state, one would ideally want, for example, to explore more complete (Ω_m, ω) or (ω_0, ω_a) surfaces and not just the extrema obtained from a visual inspection of the Planck DES data. There is also of course a range of alternative cosmological models too which, with their respective parameters, ought also to be included in any such analysis. And, as far as astrophysical assumptions are concerned, alternative density functions and galaxy morphologies are two examples of areas that deserve further consideration. Notwithstanding the simplistic approach, the findings of this chapter suggest there remains a worthwhile opportunity for further research into the suitability of the Euclid survey - in conjunction with Collett's model - as a means of constraining cosmological parameters.

Chapter 6

Gravitationally Lensed Submillimetre Galaxies

Abstract

In this chapter, we consider the implications for Collett's model of replacing the original source population based on the simulated LSST catalogue with one based on a mock catalogue of submillimetre galaxies only. In order to carry out this investigation, not only was it necessary to create a mock submillimetre galaxy catalogue, but significant elements of the coding had to be replaced or amended to recognise the format of the new data and to apply revised criteria for identifying the lensed systems. The properties of the lensed submillimetre galaxies predicted as 'discoverable' by the modified model are presented, and a comparison made with results obtained elsewhere: for reasons that remain unclear, the model is found to under-predict the *number* of lenses, although their *properties* are comparable. Therefore the results can only be considered preliminary. The chapter concludes with an investigation of the constraints imposed on different cosmologies by a source population of submillimetre galaxies. The range of cosmologies tested in this respect is necessarily limited, but for consistency follows the methodology used earlier in this project. The conclusion is that there is little sensitivity of the model to the tested cosmologies, contrary to the somewhat provocative results of Eales (2015), but several limitations in this study

have been identified and consequently there remains plenty of scope for further investigation.

6.1 Background

Submillimetre galaxies (SMGs) are distant galaxies that are particularly bright at wavelengths just below one millimetre (between infra-red and microwave radiation on the electromagnetic spectrum). A significant number of SMGs have been identified (e.g. Amvrosiadis et al. 2018) in the redshift range $2 < z < 4$ (corresponding to 10-12.5 billion years ago), with a handful found beyond $z = 5$ and only two beyond $z = 6$ (corresponding to at least 12.6 billion years ago). At these wavelengths, the light tends to come mainly from warm dust, with the heating mechanism due to a high rate of star formation. The latter may be associated with mergers: a collision between two galaxies produces a burst of star formation, with the young stellar populations in turn rapidly producing supernova that release their energy into the surrounding dust and gas. SMGs may therefore be the result of such mergers.

Since the identification of a population of faint SMGs in the 1990s (Smail et al. 1997), SMGs have come to play a significant role in the study of galaxy formation and evolution. Detection of submm radiation from distant galaxies had remained particularly elusive until then, due to the technical challenge of constructing sufficiently sensitive receivers. Atmospheric conditions also meant that observations were restricted to high mountain sites, and specific atmospheric windows. Additionally, the long wavelength of submm radiation placed a limit on spatial resolution in the absence of large filled or synthetic apertures: in 2000, the largest available apertures were in the 10-30m class, providing a spatial resolution of ~ 10 arcsecs (much coarser than optical and near-IR observations) (Blain et al. 2002).

With just one (recent) exception, to date all of the SMGs discovered at $z > 5$ have been rare examples of *extreme starburst galaxies*, with star formation rates of $> 1,000 M_{\odot}$ per year (Zavala et al. 2018). By way of comparison, a typical star formation rate is several hundred solar masses per year for sources closer to $z = 5$; the star formation rate of our own galaxy is around 1 solar mass per year. This begs the question as to whether all high-redshift SMGs are generally

as extreme as this. Alternatively, it could be argued that this is just an example of selection bias - we are simply less able to observe fainter SMGs: if we could, then we could surmise that the extremes identified so far are not in fact representative, and that the processes driving star formation have not changed substantially over the past 12.5 billion years or so. In any case, our understanding of the nature of these sources at the earliest epochs remains incomplete. To this end, gravitational lensing is a particularly powerful tool for the study of SMGs, because the amplification of light allows for the identification and study of a large population of SMGs that might otherwise be unreachable even with the current generation of detectors.

6.2 Modifications to the Model

In order to investigate the implications for the model of strongly gravitationally lensed SMGs, a number of modifications to the code were necessary.

SMG Mock Catalogue

The model in Collett (2015) was designed by default to import an existing (simulated) LSST data catalogue. However, this data does not contain information relating to SMGs. It was therefore necessary to substitute this with a catalogue of SMG data. I therefore constructed a mock SMG catalogue for this purpose based on a study by Cai et al. (2013)[1], which allows for an estimate of number counts as a function of unlensed flux (at 500 microns) and redshift. This function differs as between galaxies located below and above redshift z = 1, but for the sake of simplicity only SMGs at z > 1 have been included here. In addition to flux and redshift, the code in Collett's model requires source galaxy angular size information, and this was similarly constructed for the mock catalogue using data from Ikarashi et al. (2015, figure 6: z $\sim$ 1-3 and z > 3 plots only).

The Python code used to construct the mock SMG catalogue from the data in Cai et al. (2013) and Ikarashi et al. (2015) is shown in Appendix F.1; the mock catalogue data is stored in a *submmdata.txt* file.

[1] And in private correspondence with Z-Y Cai.

Data Import Routine

As mentioned above, the existing code in the model was designed to import an LSST catalogue. Replacing this with the mock SMG catalogue meant that SMG data would be available. However, the LSST data had been stored in a different format, namely a *pkl* file, so a further modification to the model was required to import data from the *txt* file instead. A new import routine - **loadsubmm** - has therefore been written and this has been incorporated in the *PopulationsFunctons.py* module; an example of the script is given in Appendix F.2.

Detectability Criteria

The existing code in the model ('Stage Two') applies a number of criteria to ascertain whether any particular 'idealised' lensing system is detectable. The coding default draws on the Euclid survey parameters, with selection criteria that include seeing, signal-to-noise, and magnification limits. Following the work by Negrello et al. (2010*a*), for assessing the detectability of strongly lensed SMGs the code in this stage of the model needed to be significantly modified, since the only criterion for detectability should be that the observed (lensed) flux is >100 mJy.

Two modules in particular were impacted by these changes to the selection criteria, namely *FastLensSim.py* and *SignaltoNoise.py*. After introducing a procedure for calculating and storing *lensed* flux values, a number of existing routines within these modules had either to be commented out or rewritten to avoid applying the spurious (default) criteria.

Miscellaneous Modifications

A number of relatively minor modifications were required to some of the other modules in addition to those mentioned above. In brief, these included the following: (a) the source density parameter was calculated from the data in Cai et al. (2013) to be 0.011 per sq.arcsec. (*PopulationFunctions.py*), (b) the survey area was set at 600 sq.arcsec in line with that of the Herschel Astrophysical Terahertz Large Area Survey[2] (*Surveys.py* and *MakeResults.py*), (c) the routines for scaling the total number of predictable lenses and exporting their key properties were ad-

[2] *http://herschel.cf.ac.uk*

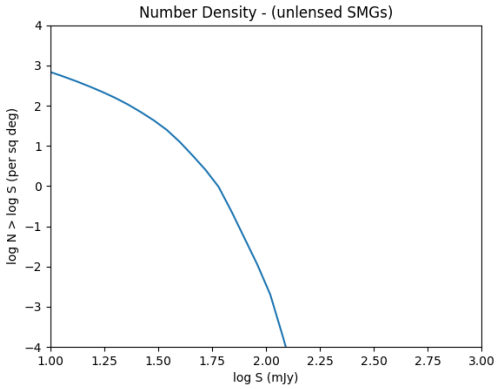

Figure 6.1: Submillimetre galaxies: Number Density (N) vs Flux (S). Based on data provided by Cai et al. (2013). The profile of the luminosity function is consistent with findings elsewhere, e.g. Negrello et al. (2010*a*). Note the data relates to *unlensed* SMGs.

justed to take into account the modified criteria (*ModelAll.py* and *MakeResults.py*).

By way of a 'sanity check', a plot of the (unlensed) SMG luminosity function based on the data from Cai et al. (2013) is shown in Figure 6.1. This plot is consistent with the findings of, for example, Negrello et al. (2010*a*).

6.3 Results

Results obtained from running the model for the SMGs simulated in the mock catalogue are displayed in Table 6.1, with key properties also plotted in Figure 6.2.

Table 6.1: SMG Predictions

Parameter	Mean	Median	Variance
Lens redshift	0.76	0.69	0.14
Source redshift	2.67	2.60	0.60
Einstein radius (arcsec)	0.84	0.80	0.18
Velocity dispersion (km/s)	224	225	2408
Magnification	7.95	6.43	25.1

With regard to redshifts, the results are broadly consistent with those of Negrello et al. (2017) where the lenses and background sources were found to have median redshifts $zl = 0.6$ and

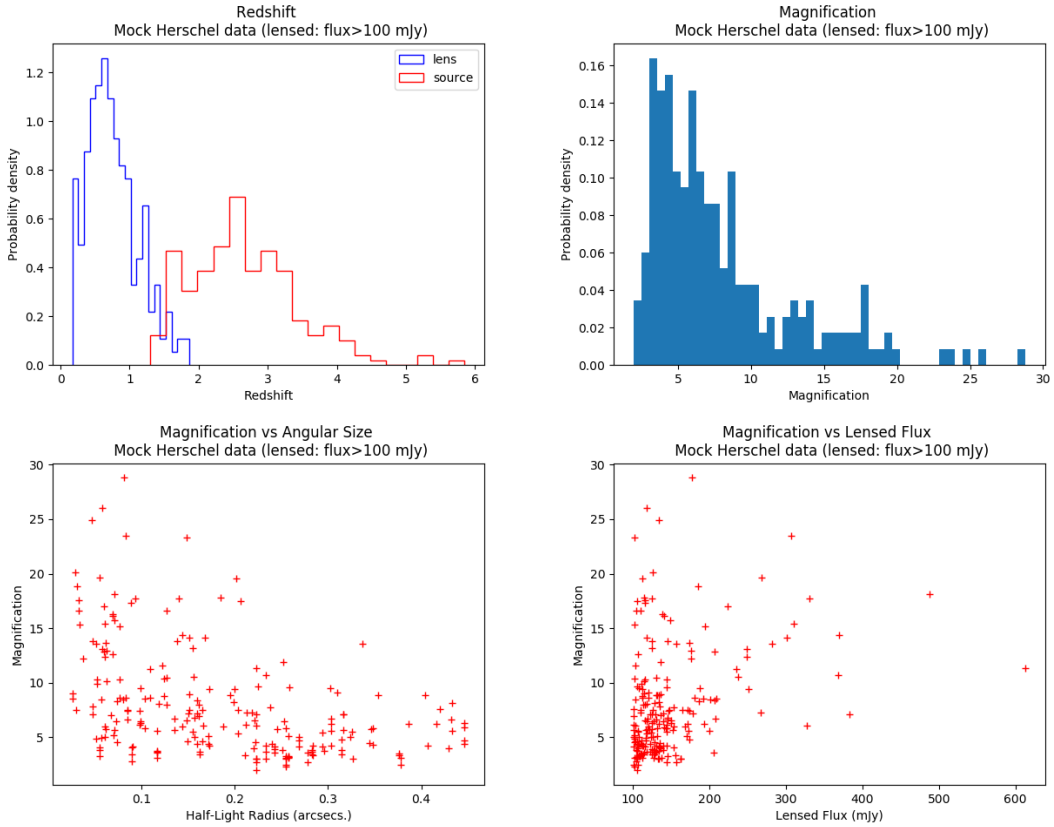

Figure 6.2: Submillimetre galaxies: redshift, magnification, flux & size properties. The predictions of the model for the properties of the SMGs are consistent with those of, for example, Negrello et al. (2017) and Bussmann et al. (2013), but by comparison the model under-predicts the *number* of lenses by a factor ∼x3; the reason for this under-prediction remains unclear.

$zs = 2.5$ respectively. The plots relating magnification to angular size, and magnification to flux are also plausible when compared to those of Bussmann et al. (2013).

The results of the model do however diverge from those we would expect when it comes to the predicted *number* of lensed galaxies. In the case of Negrello et al. (2017), a sample of 80 candidate strongly lensed SMGs with a flux density above 100 mJy at 500 μm were extracted from the Herschel Astrophysical Terahertz Large Area Survey, over an area of 600 sq.deg. But with a prediction of just 31 lensed galaxies (measured by the same criteria), Collett's model is under-predicting by a factor $\sim$x3. The reason for the under-prediction is not clear. The code was run several times with variations to the key parameters - such as pixel size, and 'postage stamp' dimensions - but the under-prediction persisted. The time available for completion of this project prohibits further investigation into this anomaly, and it remains an opportunity for future analysis accordingly, but it would appear to be a feature of the formalism of the model rather than any error in the data or coding.

Cosmological Constraints

The final part of this chapter concerns the extent to which the predictions of the model for SMGs are sensitive to the choice of cosmological parameters. In this respect, we follow a procedure similar to that described in Section 5.2.1: the modified model is run using the same four extrema of the 2-sigma (Ω_m, w) and (w_o, w_a) contours of the flat $wCDM$ and flat CPL values from Xu & Zhang (2016), and the results are then compared against the flat ΛCDM (or Concordance) cosmology for significant differences. For ease of reference, the values of the tested parameters are shown again in Figure 6.2.

Bearing in mind the under-prediction of the model, it was decided to adjust one of the variables in the code - namely, that relating to the *source plane over-density* (in the *PopulationFunctions.py* module) to increase it from its default value of 1 to 3. Adapting the model in this way raised the prediction to 85 lenses, thus effectively enforcing a match with the observations of Negrello et al. (2017).

Table 6.2: Tested Cosmologies (Planck)
source: Xu & Zhang (2016)

Cosmology	Fit	Parameters
flat ΛCDM	best-fit	$\Omega_m = 0.324, \quad h = 0.667$
flat CPL	best-fit	$\Omega_m = 0.326, \quad w_0 = -0.969, \quad w_a = 0.007, \quad h = 0.663$
flat CPL	limit 1	$\Omega_m = 0.326, \quad w_0 = -1.14, \quad w_a = 0.60, \quad h = 0.663$
flat CPL	limit 2	$\Omega_m = 0.326, \quad w_0 = -0.78, \quad w_a = -0.90, \quad h = 0.663$
flat CPL	limit 3	$\Omega_m = 0.326, \quad w_0 = -0.93, \quad w_a = 0.75, \quad h = 0.663$
flat CPL	limit 4	$\Omega_m = 0.326, \quad w_0 = -0.98, \quad w_a = 0.22, \quad h = 0.663$
flat wCDM	best-fit	$\Omega_m = 0.326, \quad w_0 = -0.964, \quad h = 0.662$
flat wCDM	limit 1	$\Omega_m = 0.312, \quad w_0 = -1.02, \quad h = 0.662$
flat wCDM	limit 2	$\Omega_m = 0.341, \quad w_0 = -0.915, \quad h = 0.662$
flat wCDM	limit 3	$\Omega_m = 0.323, \quad w_0 = -0.92, \quad h = 0.662$
flat wCDM	limit 4	$\Omega_m = 0.333, \quad w_0 = -1.01, \quad h = 0.662$

The results of running the model under the different cosmologies are shown in Figures 6.3 and 6.4, where histograms of the key properties of the lensed SMGs have been plotted against those obtained under the Concordance cosmology; this provides for a visual comparison. In Table 6.3, the differences are quantified so that a more reliable reflection of the nature of any constraints may be inferred.

Table 6.3: SMGs: EoS Likelihoods (vs Planck DES Best-Fit ΛCDM)

Cosmology		Predicted Lenses	Z-test	p_values
flat ΛCDM	best fit	85	-	-
flat CPL	best fit	85	0.7439	0.2285
flat CPL	limit 1	84	0.3721	0.3549
flat CPL	limit 2	96	1.5032	0.0664
flat CPL	limit 3	92	1.4799	0.0695
flat CPL	limit 4	91	0.9002	0.1840
flat wCDM	best fit	96	0.8439	0.1994
flat wCDM	limit 1	90	0.8496	0.1978
flat wCDM	limit 2	84	1.4225	0.0774
flat wCDM	limit 3	92	0.7190	0.2361
flat wCDM	limit 4	89	0.1456	0.4421

On a casual visual inspection, the plots in Figures 6.3 and 6.4 suggest the results of the different cosmologies *are* significantly different from those obtained under the Concordance cosmology.

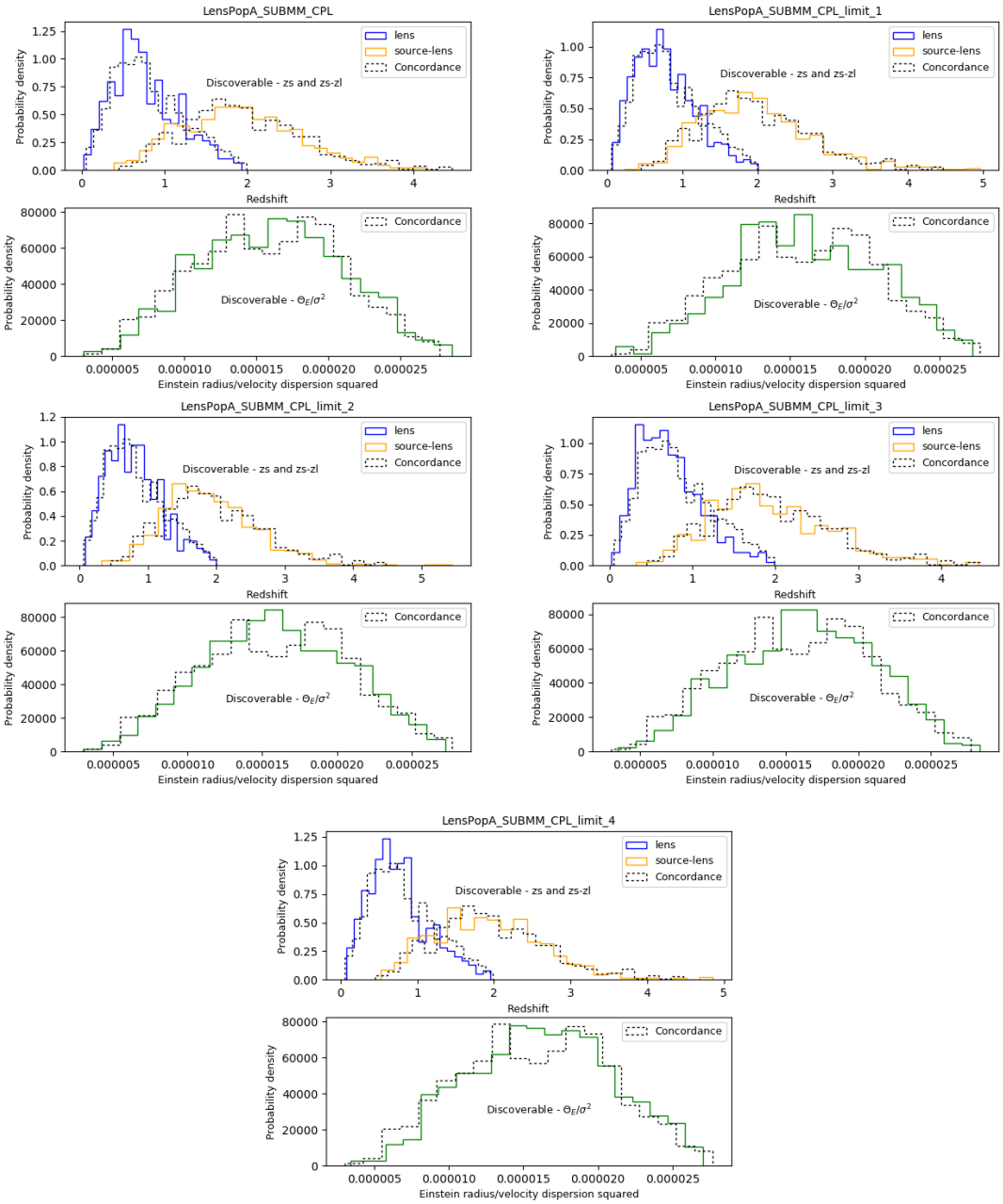

Figure 6.3: Properties of lensing systems predicted by the model under different CPL cosmologies, based on a source population comprising a mock catalogue of SMGs only. Although visually there are noticeable differences compared to predictions made under the Concordance cosmology, a quantitative analysis suggests these differences are not in fact of significance.

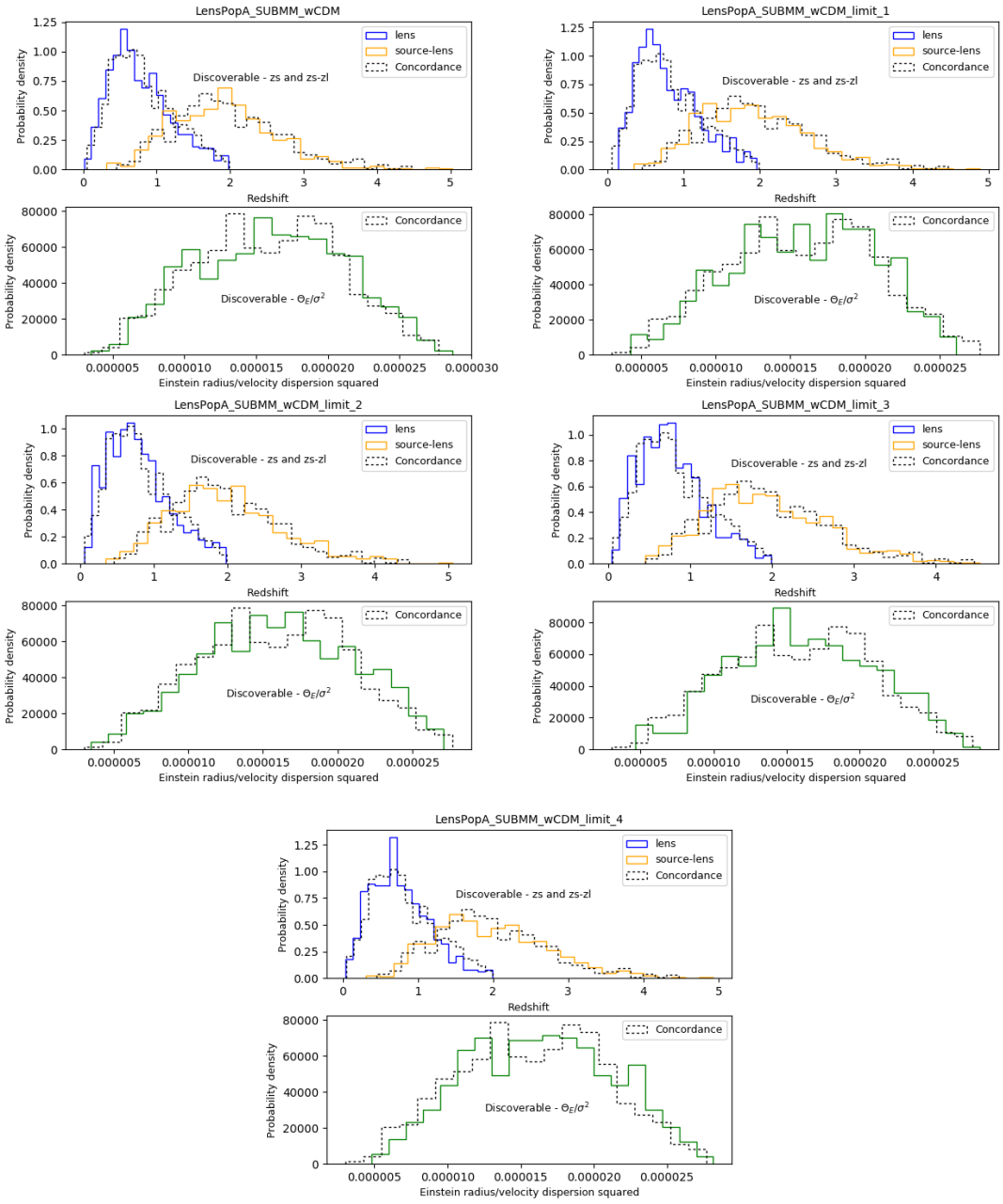

Figure 6.4: Properties of lensing systems predicted by the model under different wCDM cosmologies, based on a source population comprising a mock catalogue of SMGs only. As in the case of the CPL cosmologies illustrated in Figure 6.3, although visually there are noticeable differences compared to predictions made under the Concordance cosmology, a quantitative analysis again suggests these differences are not of significance.

However, the quantitative (and more rigorously determined) analysis in Table 6.3 indicates otherwise: the distributions of the cosmologies tested lie within about 1.5σ of the Concordance distribution, which implies there is in fact *no* such significant difference.

The conclusion of this exercise therefore is that the population of SMGs - as defined by the mock catalogue - does *not* render the model sensitive to cosmologies. We may note this is in contrast to the constraints, discussed in Section 5.2.1, that could more likely be inferred from the wider population of source galaxies imported for the original runs of the model.

6.4 Discussion - SMGs & the Model

In this chapter, we have considered an application of Collett's model (Collett 2015) to a background (ie. potential source) population comprising solely submillimetre galaxies (SMGs). The model was originally constructed to consider background galaxies based on the sky catalogues simulated for the LSST by Connolly et al. (2010), which excluded SMGs.

The objective behind replacing the source population in this way was not simply to predict the number and nature of potentially identifiable strongly lensed SMGs in future surveys, but also to ascertain whether such predictions could be used to constrain cosmological parameter values.

To carry out this analysis, the first step required the simulation of a catalogue of SMGs. This in turn meant that code had to be written specifically to identify and import relevant data, in this case from Cai et al. (2013) and Ikarashi et al. (2015). It was then necessary to implement a number of changes to some of the routines within the original model code, not least to read in data that was different in format, as well as content, to the simulated LSST catalogue. Changes to the criteria applied to identify 'discoverable' lenses were subsequently introduced into the modules, as were, finally, the calculations behind some of the lensing properties output by the model for analysis.

Whilst the *properties* of the strongly lensed SMGs predicted by the model were consistent with

expectations, the results produced an anomaly in the predicted *number* of the lensed galaxies. When compared to recent studies, such as that of Negrello et al. (2017), the predicted number is only about one third of what we might expect. It is not clear why this should be the case, and further investigation - which is beyond the present resources of this project - would be warranted, although prima facie this may be a feature of the formalism of the model rather than any error in coding or data.

With regard to constraining cosmologies, the model was run with different sets of cosmological parameters and the results compared to those obtained under the Concordance cosmology. The methodology, rationale, and parameters under test closely followed the process described in Section 5.2.1; that is, the cosmologies considered were those based on the extrema of the flat *wCDM* and flat *CPL* values from Xu & Zhang (2016)

The results of testing the cosmologies indicate that with SMGs as a source population, the model is *not* significantly sensitive to those parameter values. This is in contrast to the findings of Eales (2015), albeit for reasons that remain unclear. It should be borne in mind however that only a (necessarily) limited range of cosmologies have been tested. A further caveat is that the cause of the under-prediction by the model has not been identified, and ideally one would also want to test the model with variations both to the mock catalogue and to the astrophysical assumptions within the model. There remains plenty of scope therefore for further investigation.

Chapter 7

Conclusions & Further Work

In this project, I have considered gravitational lensing both in a historical context and as a means of furthering our understanding of modern cosmological issues. With regard to the former, I have discussed the theory behind the phenomenon, and also outlined the development of its application since the Eddington expedition of 1919 first popularised the notion that light could be deflected by gravity.

The major part of this project has been concerned with modern applications of gravitational lensing, and in particular with an analysis of the model constructed by Collett (2015) for predicting galaxy-galaxy strong gravitational lenses. In this respect, the predictions of the model have been considered primarily with regard to the forthcoming surveys by Euclid - both Wide Field and Deep Field - and additionally those of the COSMOS and WFIRST missions. Following an initial review of the coding behind the model, and several (mostly minor) modifications, predictions for the numbers of detectable lenses were obtained, as were key properties for each of the lensing systems; at present, and based on these results, the greatest number of detectable lenses appears to be that of the Wide Field Euclid survey with a count of approximately 180,000.

The overarching scientific question of this project addressed the extent to which strong gravitational lensing, as predicted by Dr Collett's model, can constrain the cosmological parameters.

The outcome of my study suggests that a meaningful constraint on the density parameter Ω_m is possible, albeit somewhat weak and not as constraining as one might expect from work elsewhere (e.g. Eales 2015). Surprisingly, the constraints do not appear particularly sensitive to the astrophysical assumptions within the model.

For the penultimate chapter of this project, the model was adapted to examine the consequences of a background source population comprising submillimetre galaxies (SMGs), in place of the simulated LSST catalogue of galaxies on which it was initially based. Predictions were obtained for strong gravitationally lensed SMGs following a methodology similar to that used by Negrello et al. (2010*a*), and an analysis also carried out to establish the degree to which those predictions would be sensitive to different cosmologies. The conclusion with regard to the latter is that as far as SMGs are concerned, there is *no* significant sensitivity to the tested cosmologies.

Whilst Dr Collett's model has already proved a valuable tool for analyses such as those carried out in this project, steps to investigate and resolve the limitations identified along the way stand to improve substantially its usefulness and reliability for future studies of gravitational lensing. In this respect, recent technological and intellectual advances have led to a rapid growth in the importance and applicability of gravitational lensing to current cosmological and astronomical research, and as a consequence there exists a multitude of opportunities where such studies are likely to be of considerable value. I have touched on a number of these already, but it is useful now to look ahead and highlight some of those that may be linked directly to the topics covered in the project.

Firstly, the analyses carried out here have mostly involved the application of the model in Collett (2015) to source galaxies provided by a simulated LSST catalogue. There are limitations to the depth of this catalogue, and these suggest the model may be under-predicting the number of high magnification events that Euclid will actually detect. The Euclid surveys, when they take place, are therefore likely to identify suitable follow-up targets for pointed observations by facilities such as ELT in 2025, which in turn will provide an opportunity for studies of stellar formation that have hitherto not been possible.

Another shortfall in the LSST source catalogue is the absence of submillimetre galaxies. Within this project, I have made use of data provided by Cai et al. (2013) in order to create a mock catalogue of such galaxies, and also amended the code within the model to make use of this data. Expanding the source catalogue in this way should lead to a more complete application of the lensing model, and furthermore the mock catalogue would now be available for use elsewhere if required. However, the catalogue remains incomplete in that, for example, it does not include quasars or ultra-high-redshift galaxies. There remains therefore an opportunity to adapt the source catalogue further in order to make use of a more realistic source dataset, and consequently obtain more reliable predictions from the model.

In the case of the SMGs, a particular area amenable to further investigation is the unanswered question as to why the model appears to be under-predicting the number of detectable lenses, when compared to other studies (e.g. Negrello et al. 2017). It is to some extent reassuring that the profiles of the detectable lensing systems are consistent with those predicted elsewhere, but an unexplained under-prediction in the number of these by a factor $\sim$x3 necessarily raises a concern over the integrity of the model when applied to a source population of SMGs.

With regard to surveys other than that of Euclid, I have discussed the model's predictions for WFIRST and also commented on the limitations inherent in the assumptions behind it. In short, and at the very least, there remain opportunities to improve on the estimates for some of the parameter values within the model for the WFIRST survey, and in particular there is a need to modify the model to include *all* the filter bands rather than the single J_129 filter adopted here.

On the subject of constraining cosmologies, the analysis in this area has necessarily been simplistic given the scope of this project. Consequently, however, the study undertaken clearly points to opportunities for strengthening the model's role in this respect. An obvious example is the need to consider the implications for a wider range of combinations of cosmological parameter values than those tested here (which were limited to just a few EoS and Ω extrema). No less importantly, a more varied set of astrophysics assumptions (e.g. luminosity functions) could also

be built into, and tested within, the model. Moreover, I believe there is scope for subjecting the comparisons between predictions of the different cosmologies to a more sophisticated statistical analysis, rather than the (unavoidably) brute application of a Chi-Square measure used in this project. Subject to these refinements, the model in my opinion has a valuable role to play in constraining many of the cosmological parameters. This is particularly so given too that it could be used *alongside* other techniques, such as the time-delay methods explored by the H0LiCOW project. This serves once again to highlight the opportunities and benefits to be derived from further work on the topics raised and, I hope, appropriately covered in this project.

Appendices

Appendix A

Structure of the Model

A.1 Source Codes

The *original* source codes relating to the modules referred to in this project are shown below, in the following order (the abbreviations are those used throughout this document):-

- MakeLensPop.py (MLP)
- PopulationFunctions.py (PFs)
- distances.py (Dis)
- ModelAll.py (MAll)
- FastLensSim.py (FLS)
- SBModels.py (SBM)
- SBProfiles.py (SBP)
- Surveys.py (Sur)
- StochasticObserving.py (Sto)
- SignaltoNoise.py (SN)
- MakeResults.py (MRs)

It should be emphasised that these modules and other related Python code are the result of work by Collett (2015); at the time of writing these are available for download from *https://github.com/tcollett/LensPop*.

```
import distances
from scipy import interpolate
import cPickle,numpy,math
import indexTricks as iT
import pylab as plt
from PopulationFunctions import *

class LensPopulation(LensPopulation_):
    def  __init__(self,zlmax=2,sigfloor=250,D=None,reset=True,

bands=['F814W_ACS','g_SDSS','r_SDSS','i_SDSS','z_SDSS','Y_UKIRT','VIS']
                  ): #sadface
        self.sigfloor=sigfloor
        self.zlmax=zlmax
        self.bands=bands

        self.beginRedshiftDependentRelation(D,reset)
        self.beginLensPopulation(D,reset)

    def phi(self,sigma,z):
    #you can change this, but remember to reset the splines if you do.
        sigma[sigma==0]+=1e-6
        phi_star=(8*10**-3)*self.D.h**3
        alpha=2.32
        beta=2.67
        sigst=161
        phi=phi_star * \
            ((sigma*1./sigst)**alpha)*\
            numpy.exp(-(sigma*1./sigst)**beta)*beta/\
            math.gamma(alpha*1./beta)/\
            (1.*sigma)

        #phi*=(1+z)**(-2.5)
        self.nozdependence=True

        return phi


class SourcePopulation(SourcePopulation_):
    def  __init__(self,D=None,reset=False,

bands=['F814W_ACS','g_SDSS','r_SDSS','i_SDSS','z_SDSS','Y_UKIRT'],
population="cosmos"
                  ):
        self.bands=bands
        self.beginRedshiftDependentRelation(D,reset)
        if population=="cosmos":
            self.loadcosmos()
        elif population=="lsst":
```

```
47              self.loadlsst()
48
49
50          #NB all the functions are in the inherited from class.
51
52  #,'VIS'
53
54  #=======================================
55  class LensSample():
56      """
57      Wrapper for all the other objects so you can just call it, and then
    …  run
58      Generate_Lens_Pop to get a fairly drawn lens population
59      """
60      def  __init__(self,D=None,reset=False,zlmax=2,sigfloor=100,
61
    …  bands=['F814W_ACS','g_SDSS','r_SDSS','i_SDSS','z_SDSS','Y_UKIRT'],cosmo=
    …  [0.3,0.7,0.7],sourcepop="lsst"
62                      ):
63          self.sourcepopulation=sourcepop
64          if D==None:
65              import distances
66              D=distances.Distance(cosmo=cosmo)
67
68
    …  self.L=LensPopulation(reset=reset,sigfloor=sigfloor,zlmax=zlmax,bands=
    …  bands,D=D)
69
70
    …  self.S=SourcePopulation(reset=reset,bands=bands,D=D,population=sourcepop
    …  )
71
72          self.E=EinsteinRadiusTools(D=D)
73
74      def Lenses_on_sky(self):
75          self.ndeflectors=self.L.Ndeflectors(self.L.zlmax)
76          return self.ndeflectors
77
78      def
    …  Generate_Lens_Pop(self,N,firstod=1,nsources=1,prunenonlenses=True,save=
    …  True):
79          import time
80          t0=time.clock()
81          if prunenonlenses==False: assert N<60000
82
83          self.lens={}
84          self.reallens={}
85          M=N*1
86          l=-1
```

```
87          l2=-1
88          while M>0:
89              timeleft="who knows"
90              if M!=N:
91                  tnow=time.clock()
92                  ti=(tnow-t0)/float(N-M)
93                  timeleft=ti*M/60.
94
95
96              print M,timeleft," minutes left"
97              if M>100000:
98                  n=100000
99              else:
100                 n=M*1
101             M-=n
102             zl,sigl,ml,rl,ql=self.L.drawLensPopulation(n)
103
…   zs,ms,xs,ys,qs,ps,rs,mstar,mhalo=self.S.drawSourcePopulation(n*nsources,
…   sourceplaneoverdensity=firstod,returnmasses=True)
104
105             zl1=zl*1
106             sigl1=sigl*1
107             for i in range(nsources-1):
108                 zl=numpy.concatenate((zl,zl1))
109                 sigl=numpy.concatenate((sigl,sigl1))
110
111             b=self.E.sie_rein(sigl,zl,zs)
112             for i in range(n):
113                 l +=1
114                 self.lens[l]={}
115                 if b[i]**2>(xs[i]**2+ys[i]**2):
116                     self.lens[l]["lens?"]=True
117                 else:
118                     self.lens[l]["lens?"]=False
119
120                 self.lens[l]["b"]={}
121                 self.lens[l]["zs"]={}
122                 self.lens[l]["zl"]=zl[i]
123                 self.lens[l]["sigl"]=sigl[i]
124                 for j in range(nsources):
125                     self.lens[l]["zs"][j+1]=zs[i+j*n]
126                     self.lens[l]["b"][j+1] =b[i+j*n]
127
128                 self.lens[l]["ml"]={}
129                 self.lens[l]["rl"]={}
130                 self.lens[l]["ms"]={}
131
132                 for band in ml.keys():
133                         self.lens[l]["ml"][band]=ml[band][i]
```

```
134                             self.lens[l]["rl"][band]=rl[band][i]
135                 self.lens[l]["ql"]=ql[i]
136 
137                 self.lens[l]["ms"]={}
138                 self.lens[l]["xs"]={}
139                 self.lens[l]["ys"]={}
140                 self.lens[l]["rs"]={}
141                 self.lens[l]["qs"]={}
142                 self.lens[l]["ps"]={}
143                 self.lens[l]["mstar"]={}
144                 self.lens[l]["mhalo"]={}
145 
146                 for j in range(nsources):
147                     self.lens[l]["ms"][j+1]={}
148                     for band in ml.keys():
149                         self.lens[l]["ms"][j+1][band]=ms[band][i+j*n]
150                     self.lens[l]["zs"][j+1]=zs[i+j*n]
151                     self.lens[l]["b"][j+1] =b[i+j*n]
152                     self.lens[l]["xs"][j+1]=xs[i+j*n]
153                     self.lens[l]["ys"][j+1]=ys[i+j*n]
154                     self.lens[l]["rs"][j+1]=rs[i+j*n]
155                     self.lens[l]["qs"][j+1]=qs[i+j*n]
156                     self.lens[l]["ps"][j+1]=ps[i+j*n]
157                     self.lens[l]["mhalo"][j+1]=mstar[i+j*n]
158                     self.lens[l]["mstar"][j+1]=mhalo[i+j*n]
159 
160 
161                 if self.lens[l]["lens?"]:
162                     if prunenonlenses:
163                         l2+=1
164 
165                         self.reallens[l2]=self.lens[l].copy()
166 
167                         del self.lens
168                         self.lens={}
169 
170                         if l2%1000==0:
171                             print l2
172 
173                         if (l2+1)%10000==0:
174                           if save:
175 
…fn="idealisedlenses/lenspopulation_%s_%i.pkl"%(self.sourcepopulation,l2-
…10000+1)
176                             print fn
177                             f=open(fn,'wb')
178                             cPickle.dump(self.reallens,f,2)
179                             f.close()
180                             del self.reallens
```

```
181                             self.reallens={}
182
183                 elif prunenonlenses:
184                     del self.lens
185                     self.lens={}
186         if save:
187
…  fn="idealisedlenses/lenspopulation_%s_residual_%i.pkl"%(self.
…  sourcepopulation,l2)
188             print l2,fn
189             f=open(fn,'wb')
190             cPickle.dump(self.reallens,f,2)
191             f.close()
192
193         if prunenonlenses==False:
194           if save:
195
…  f=open("idealisedlenses/nonlenspopulation_%s.pkl"%self.sourcepopulation,
…  'wb')
196             cPickle.dump(self.lens,f,2)
197             f.close()
198             print len(self.lens.keys())
199
200         self.lens=self.reallens
201
202     def LoadLensPop(self,j=0,sourcepopulation="lsst"):
203
…  f=open("idealisedlenses/lenspopulation_%s_%i.pkl"%(sourcepopulation,j),'
…  rb')
204         self.lens=cPickle.load(f)
205         f.close()
206
207
208     def Pick_a_lens(self,i=None,dspl=False,tspl=False):
209         if i ==None:
210             numpy.random.randint(0,self.n)
211
212         self.rli={}
213         self.mli={}
214         self.msi={}
215         self.msi2={}
216         self.msi3={}
217
218         for band in self.L.bands:
219             self.rli[band]=self.rl[band][i]
220             self.mli[band]=self.ml[band][i]
221         for band in self.S.bands:
222             self.msi[band]=self.ms[band][i]
223             if dspl or tspl:
```

```
224                 self.msi2[band]=self.ms2[band][i]
225                 if tspl:self.msi3[band]=self.ms3[band][i]
226
227
…  preselection=self.apply_preselection(self.mli["i_SDSS"],self.zl[i])
228         if dspl==False and tspl==False:
229             return
…  [self.mli,self.rli,self.ql[i],self.bl[i]],[self.msi,self.xs[i],self.yl[i
…  ],self.qs[i],self.ps[i],self.rs[i]],[self.zl[i],self.zs[i]],preselection
230         elif tspl==False:
231             return
…  [self.mli,self.rli,self.ql[i],self.bl[i]],[self.msi,self.xs[i],self.yl[i
…  ],self.qs[i],self.ps[i],self.rs[i]],[self.bl2[i],self.msi2,self.xs2[i],
…  self.yl2[i],self.qs2[i],self.ps2[i],self.rs2[i]],[self.zl[i],self.zs[i],
…  self.zs2[i],self.sigl[i],self.Mvs[i],self.r_phys[i]],preselection
232
233         else:
234             return [self.mli,self.rli,self.ql[i],self.bl[i]],
…  [self.msi,self.xs[i],self.yl[i],self.qs[i],self.ps[i],self.rs[i]],
…  [self.bl2[i],self.msi2,self.xs2[i],self.yl2[i],self.qs2[i],self.ps2[i],
…  self.rs2[i]],
…  [self.bl3[i],self.msi3,self.xs3[i],self.yl3[i],self.qs3[i],self.ps3[i],
…  self.rs3[i]],
…  [self.zl[i],self.zs[i],self.zs2[i],self.zs3[i],self.sigl[i],self.Mvs[i],
…  self.r_phys[i]],     preselection
235
236
237     def apply_preselection(self,imag,z):
238         if imag<15: return False
239         if imag>23:return False
240         if z<0.05: return False
241         return True
242
243 if __name__ == "__main__":
244     import distances
245     fsky=1
246     D=distances.Distance()
247     Lpop=LensPopulation(reset=True,sigfloor=100,zlmax=2,D=D)
248     Ndeflectors=Lpop.Ndeflectors(2,zmin=0,fsky=1)
249
…  L=LensSample(reset=False,sigfloor=100,cosmo=[0.3,0.7,0.7],sourcepop="
…  lsst")
250
…  L.Generate_Lens_Pop(int(Ndeflectors),firstod=1,nsources=1,prunenonlenses
…  =True)
251
```

```
import distances
from scipy import interpolate
import cPickle,numpy,math
import indexTricks as iT

#=============================================================================
==============

class RedshiftDependentRelation():
    def __init__(self,D=None,reset=False,cosmo=[0.3,0.7,0.7]):
        self.beginRedshiftDependentRelation(D,reset=reset,cosmo=cosmo)

    def
beginRedshiftDependentRelation(self,D,reset,zmax=10,cosmo=[0.3,0.7,0.7])
:
        self.zmax=zmax
        self.zbins,self.dz=numpy.linspace(0,self.zmax,401,retstep=True)

self.z2bins,self.dz2=numpy.linspace(0,self.zmax,201,retstep=True)
        if D==None:
            import distances
            D=distances.Distance(cosmo=cosmo)
        self.D=D

        if reset!=True:
            try:
            #load useful redshift splines
                splinedump=open("redshiftsplines.pkl","rb")

self.Da_spline,self.Dmod_spline,self.volume_spline,self.Da_bispline=
cPickle.load(splinedump)
            except IOError or EOFError:
                self.redshiftfunctions()
        else:
            self.redshiftfunctions()

    def redshiftfunctions(self):
        D=self.D
        zbins=self.zbins
        z2bins=self.z2bins
        Dabins=zbins*0.0
        Dmodbins=zbins*0.0
        Da2bins=numpy.zeros((z2bins.size,z2bins.size))
        volumebins=zbins*0.0
        for i in range(zbins.size):
            Dabins[i]=D.Da(zbins[i])
            Dmodbins[i]=D.distance_modulus(zbins[i])
            volumebins[i]=D.volume(zbins[i])
        for i in range(z2bins.size):
```

```
44              for j in range(z2bins.size):
45                  if j>i:
46                      Da2bins[i,j]=D.Da(z2bins[i],z2bins[j])
47
48          self.Da_spline=interpolate.splrep(zbins,Dabins)
49          self.Dmod_spline=interpolate.splrep(zbins,Dmodbins)
50
51          self.volume_spline=interpolate.splrep(zbins,volumebins)
52
53          z2d=iT.coords((z2bins.size,z2bins.size))*self.dz2
54
… self.Da_bispline=interpolate.RectBivariateSpline(z2bins,z2bins,Da2bins)
55
56          #pickle the splines
57          splinedump=open("redshiftsplines.pkl","wb")
58
… cPickle.dump([self.Da_spline,self.Dmod_spline,self.volume_spline,self.
… Da_bispline],splinedump,2)
59
60      def Volume(self,z1,z2=None):
61          if z2==None:
62              return self.splev(z1,self.volume_spline)
63          else:
64              z1,z2=self.biassert(z1,z2)
65              return
… self.splev(z2,self.volume_spline)-self.splev(z1,self.volume_spline)
66
67      def Da(self,z1,z2=None,units="Mpc"):
68          if units=="kpc":
69              corfrac=1000
70          elif units=="Mpc":
71              corfrac=1
72          else:
73              print "don't know those units yet"
74          if z2==None:
75              return self.splev(z1,self.Da_spline)*corfrac
76          else:
77              z1,z2=self.biassert(z1,z2)
78              return self.Da_bispline.ev(z1,z2)*corfrac
79
80      def Dmod(self,z):
81          return self.splev(z,self.Dmod_spline)
82
83      def splev(self,x,f_of_x_as_spline):
84          return interpolate.splev(x,f_of_x_as_spline)
85
86      def bisplev(self,x,y,f_ofxy_as_bispline):
87          return interpolate.bisplev(x,y,f_ofxy_as_bispline)
88
```

```
89      def biassert(self,z1,z2):
90              try: len(z1)
91              except TypeError:z1=[z1]
92              try: len(z2)
93              except TypeError:z2=[z2]
94              if len(z1)==1 and len(z2)!=1:z1=numpy.ones(len(z2))*z1[0]
95              if len(z2)==1 and len(z1)!=1:z2=numpy.ones(len(z1))*z2[0]
96              assert len(z1)==len(z2),"get it together"
97              return z1,z2
98
99  #=============================================================================
…   ==============
100
101
102 class EinsteinRadiusTools(RedshiftDependentRelation):
103     def  __init__(self,D=None,reset=False):
104         self.beginRedshiftDependentRelation(D,reset)
105         self.c=299792
106
107     def sie_sig(self,rein,zl,zs):
108         self.c=299792
109         ds=self.Da(zs)
110         dls=self.Da(zl,zs)
111         sig=(rein*(ds*self.c**2)/(206265*4*math.pi*dls))**0.5
112         return sig
113     def sie_rein(self,sig,zl,zs):
114         self.c=299792
115         ds=self.Da(zs)
116         dls=self.Da(zl,zs)
117         rein=sig**2*((ds*self.c**2)/(206265*4*math.pi*dls))**-1
118         rein[rein<0]=0
119         return rein
120
121
122 #=============================================================================
…   ==============
123 class Population(RedshiftDependentRelation):
124     def  __init__(self):
125         pass
126
127     def draw_apparent_magnitude(self,M,z,band=None,colours=None):
128         if band!=None:
129             colours=self.colour(z,band)
130         if colours==None:
131             colours=0
132             print "warning no k-correction"
133         Dmods=self.Dmod(z)
134         ml = M - colours + Dmods
135         return ml
```

```
136
137     def draw_apparent_size(self,r_phys,z):
138         rl = r_phys/(self.Da(z,units="kpc"))
139         rl *= 206264
140         return rl
141
142 #=============================================================================
    ==============
143
144 class LensPopulation_(Population):
145     def  __init__(self,zlmax=2,sigfloor=100,D=None,reset=True,
146
    bands=['F814W_ACS','g_SDSS','r_SDSS','i_SDSS','z_SDSS','Y_UKIRT','VIS'],
    cosmo=[0.3,0.7,0.7]
147                   ): #sadface
148         self.sigfloor=sigfloor
149         self.zlmax=zlmax
150         self.bands=bands
151
152         self.beginRedshiftDependentRelation(D,reset)
153         self.beginLensPopulation(D,reset)
154
155
156     def beginLensPopulation(self,D,reset):
157         reset=True
158         if reset!=True:
159             try:
160             #load Lens-population splines
161                 splinedump=open("lenspopsplines.pkl","rb")
162
    self.cdfdNdzasspline,self.cdfdsigdzasspline,self.dNdzspline,self.zlbins,
    zlmax,sigfloor,self.colourspline,bands=cPickle.load(splinedump)
163             except IOError or EOFError or ValueError:
164                 self.lenspopfunctions()
165             #check sigfloor and zlmax are same as requested
166             if zlmax!=self.zlmax or self.sigfloor!=sigfloor:
167                 self.lenspopfunctions()
168             #check all the necessary colours are included
169             redocolours=False
170             for band in self.bands:
171                 if band not in bands:redocolours=True
172             if redocolours:
173                 self.Colourspline()
174                 self.lensPopSplineDump()
175         else:
176             self.lenspopfunctions()
177
178     def lenspopfunctions(self):
179         self.Psigzspline()
```

```
180         self.Colourspline()
181         self.lensPopSplineDump()
182 
183     def Psigzspline(self):
184         #"""
185         #drawing from a 2d pdf is a pain; should probably make this into
…   its own module
186 
…   self.zlbins,self.dzl=numpy.linspace(0,self.zlmax,201,retstep=True)
187         sigmas=numpy.linspace(self.sigfloor,400,401)
188         self.sigbins=sigmas
189         dNdz=self.zlbins*0
190         Csiggivenz=numpy.zeros((sigmas.size,self.zlbins.size))
191         CDFbins=numpy.linspace(0,1,1001)
192         siggivenCz=numpy.zeros((CDFbins.size,self.zlbins.size))
193         for i in range(len(self.zlbins)):
194             z=self.zlbins[i]
195             dphidsiggivenz=self.phi(sigmas,z)
196             phisigspline=interpolate.splrep(sigmas,dphidsiggivenz)
197             tot=interpolate.splint(self.sigfloor,500,phisigspline)
198 
…   Csiggivenz[:,i]=numpy.cumsum(dphidsiggivenz)/numpy.sum(dphidsiggivenz)
199             Csiggivenzspline=interpolate.splrep(Csiggivenz[:,i],sigmas)
200             siggivenCz[:,i]=interpolate.splev(CDFbins,Csiggivenzspline)
201             if z!=0:
202 
…   dNdz[i]=tot*(self.Volume(z)-self.Volume(z-self.dzl))/self.dzl
203 
204         Nofzcdf=numpy.cumsum(dNdz)/numpy.sum(dNdz)
205         #import pylab as plt
206         #plt.plot(self.zlbins,Nofzcdf)
207         #plt.show()
208         #exit()
209         self.cdfdNdzasspline=interpolate.splrep(Nofzcdf,self.zlbins)
210 
211         self.dNdzspline=interpolate.splrep(self.zlbins,dNdz)
212         N=interpolate.splint(0,self.zlmax,self.dNdzspline)
213 
214         self.cdfdsigdzasspline=interpolate.RectBivariateSpline(\
215             CDFbins,self.zlbins,siggivenCz)
216 
217         dphidsiggivenz0=self.phi(sigmas,sigmas*0)
218         cdfdNdsigz0=dphidsiggivenz0.cumsum()/dphidsiggivenz0.sum()
219         self.cdfdNdsigz0asspline=interpolate.splrep(cdfdNdsigz0,sigmas)
220 
221 
222         #"""
223         #phi is redshift independant.
224 
```

```
225
226
227     def Colourspline(self):
228         from stellarpop import tools
229         sed = tools.getSED('BC_Z=1.0_age=10.00gyr')
230         #different SEDs don't change things much
231
232         rband=tools.filterfromfile('r_SDSS')
233         z=self.zlbins
234         self.colourspline={}
235         for band in self.bands:
236           if band!="VIS":
237             c=z*0
238             Cband=tools.filterfromfile(band)
239             for i in range(len(z)):
240                 c[i] = - (tools.ABFM(Cband,sed,z[i]) -
…  tools.ABFM(rband,sed,0))
241             self.colourspline[band]=interpolate.splrep(z,c)
242
243
244     def lensPopSplineDump(self):
245         splinedump=open("lenspopsplines.pkl","wb")
246
…  cPickle.dump([self.cdfdNdzasspline,self.cdfdNdsigz0asspline,self.
…  cdfdsigdzasspline,self.dNdzspline,self.zlbins,self.zlmax,self.sigfloor,
…  self.colourspline,self.bands],splinedump,2)
247
248     def draw_z(self,N):
249         return
…  interpolate.splev(numpy.random.random(N),self.cdfdNdzasspline)
250
251     def draw_sigma(self,z):
252         try: len(z)
253         except TypeError:z=[z]
254         if self.nozdependence:
255             sigs
…  =interpolate.splev(numpy.random.random(len(z)),self.cdfdNdsigz0asspline)
256             return sigs
257         else:
258             print "Warning: drawing from 2dpdf is low accuracy"
259             return
…  self.cdfdsigdzasspline.ev(numpy.random.random(len(z)),z)
260
261     def draw_zsig(self,N):
262         z=self.draw_z(N)
263         sig=self.draw_sigma(z)
264         return z,sig
265
266     def EarlyTypeRelations(self,sigma,z=None,scatter=True,band=None):#z
```

```
266… dependence not encoded currently
267         #Hyde and Bernardi, M = r band absolute magnitude.
268         V=numpy.log10(sigma)
269         Mr=(-0.37+(0.37**2-(4*(0.006)*(2.97+V)))**0.5)/(2*0.006)
270         if scatter:
271             Mr+=numpy.random.randn(len(Mr))*(0.15/2.4)
272 
273         #R=4.72+0.63*Mr+0.02*Mr**2 #rest-frame R_band size.
274         R=2.46-2.79*V+0.84*V**2
275         if scatter:
276             R+=numpy.random.randn(len(R))*0.11
277 
278         #convert to observed r band size;
279         r_phys = 10**R
280 
281         return Mr,r_phys
282 
283     def colour(self,z,band):
284         return interpolate.splev(z,self.colourspline[band])
285 
286     def Ndeflectors(self,z,zmin=0,fsky=1):
287         if zmin>z:
288             z,zmin=zmin,z
289         N=interpolate.splint(zmin,z,self.dNdzspline)
290         N*=fsky
291         return N
292 
293     def phi(self,sigma,z):
294         sigma[sigma==0]+=1e-6
295         phi_star=(8*10**-3)*self.D.h**3
296         alpha=2.32
297         beta=2.67
298         sigst=161
299         phi=phi_star * \
300             ((sigma*1./sigst)**alpha)*\
301             numpy.exp(-(sigma*1./sigst)**beta)*beta/\
302             math.gamma(alpha*1./beta)/\
303             (1.*sigma)
304 
305         phi*=(1+z)**(-2.5)
306         return phi
307 
308     def draw_flattening(self,sigma,z=None):
309         x=sigma
310         y=0.378-0.000572*x
311         e=numpy.random.rayleigh(y)
312         q=1-e
313         #dont like ultraflattened masses:
314         while len(q[q<0.2])>0 or len(q[q>1])>0:
```

```
315             q[q<0.2]=1-numpy.random.rayleigh(y[q<0.2])
316             q[q>1]=1-numpy.random.rayleigh(y[q>1])
317         return q
318 
319     def drawLensPopulation(self,number):
320         self.zl,self.sigl=self.draw_zsig(number)
321         self.ql=self.draw_flattening(self.sigl)
322 
…   self.Mr,self.r_phys_nocol=self.EarlyTypeRelations(self.sigl,self.zl,
…   scatter=True)
323         self.ml={}
324         self.rl={}
325         self.r_phys={}
326         for band in self.bands:
327             self.r_phys[band]=self.r_phys_nocol#could add a colorfunc
…   here
328             if band !="VIS":
329 
…   self.ml[band]=self.draw_apparent_magnitude(self.Mr,self.zl,band)
330             else: pass
331 
…   self.rl[band]=self.draw_apparent_size(self.r_phys[band],self.zl)
332         return self.zl,self.sigl,self.ml,self.rl,self.ql
333 
334 #=============================================================================
…   ==============
335 
336 class SourcePopulation_(Population):
337     def  __init__(self,D=None,reset=False,
338 
…   bands=['F814W_ACS','g_SDSS','r_SDSS','i_SDSS','z_SDSS','Y_UKIRT','VIS'],
…   cosmo=[0.3,0.7,0.7],population="cosmos"
339                   ):
340         self.bands=bands
341 
342         self.beginRedshiftDependentRelation(D,reset)
343 
344         if population=="cosmos":
345             self.loadcosmos()
346         elif population=="lsst":
347             self.loadlsst()
348 
349     def loadcosmos(self):
350         self.population="cosmos"
351 
352         try:
353             #load pickledcosmos
354             cosmosdump=open("cosmosdata.pkl","rb")
355             cosmosphotozs=cPickle.load(cosmosdump)
```

```
356         except IOError or EOFError:
357             import re
358 
…  photozs=open('../Forecaster/cosmos_zphot_mag25.tbl','r').readlines()[10:
…  ]
359             splinedump=open("cosmosdata.pkl","wb")
360             cols=len(re.split(r"\s+",photozs[0])[1:-1])
361             rows=len(photozs)
362             cosmosphotozs=numpy.empty((cols,rows))
363             for i in range(len(photozs)):
364                 line=photozs[i]
365                 l=numpy.array(re.split(r"\s+",line)[1:-1])
366                 l[l=='null']=999
367                 cosmosphotozs[:,i]=l
368             cosmosphotozs=cosmosphotozs.astype(numpy.float)
369             raz=cosmosphotozs[2,:]
370             decz=cosmosphotozs[3,:]
371             zc=cosmosphotozs[6,:]
372             cosmosphotozs=cosmosphotozs[:,((zc<10)&(zc>0))]
373             cPickle.dump(cosmosphotozs,splinedump,2)
374 
375         self.zc=cosmosphotozs[6,:]
376 
377         self.m={}
378         index={}
379         index["g_SDSS"]=23 #lets pretend sdss_g=cfht_g etc
380         index["r_SDSS"]=24
381         index["i_SDSS"]=25
382         index["z_SDSS"]=26
383         index["Y_UKIRT"]=27 #pretend Y_DES=ic whatever ic is...
384         index["F814W_ACS"]=25 # But we'll make do with F814==i
385 
386         for band in self.bands:
387           if band!="VIS":
388             self.m[band]=cosmosphotozs[index[band],:]
389 
…  self.m["VIS"]=(self.m["r_SDSS"]+self.m["i_SDSS"]+self.m["z_SDSS"])/3
390 
391         self.Mv=cosmosphotozs[-1,:]
392 
393         self.mstar=cosmosphotozs[-1,:]*0.
394         self.mhalo=cosmosphotozs[-1,:]*0.
395 
396     def loadlsst(self):
397         self.population="lsst"
398         import cPickle
399 
400         f=open('lsst.1sqdegree_catalog2.pkl','rb')
401         print "new lsst catalogue"
```

```
402         data=cPickle.load(f)
403         f.close()
404 
405         self.zc=data[:,2]
406         self.m={}
407         #print data[:,0].max()-data[:,0].min()
408         #print data[:,1].max()-data[:,1].min()
409 
410         self.m["g_SDSS"]=data[:,3]
411         self.m["r_SDSS"]=data[:,4]
412         self.m["i_SDSS"]=data[:,5]
413         self.m["z_SDSS"]=data[:,6]
414         self.m["F814W_ACS"]=data[:,5] # we'll make do with F814==i
415         self.m["Y_UKIRT"]=data[:,6]*99 #there is no Y band data atm
416         self.mstar=data[:,12]
417         self.mhalo=data[:,13]
418 
…   self.m["VIS"]=(self.m["r_SDSS"]+self.m["i_SDSS"]+self.m["z_SDSS"])/3
419         self.Mv=data[:,7]
420 
421     def RofMz(self,M,z,scatter=True,band=None):#band independent so far
422     #{mosleh et al}, {Huang, Ferguson et al.}, Newton SLACS XI.
423         r_phys=((M/-19.5)**-0.22)*((1.+z)/5.)**(-1.2)
424         # is the same as
425         R=-(M+18.)/4.
426         r_phys=(10**R)*((1.+z)/1.6)**(-1.2)
427 
428         if scatter!=False:
429             if scatter==True:scatter=0.35 #dex
430             self.scattered=10**(numpy.random.randn(len(r_phys))*scatter)
431             r_phys*=self.scattered
432 
433         return r_phys
434 
435 
436     def draw_flattening(self,N):
437         y=numpy.ones(N*1.5)*0.3
438         e=numpy.random.rayleigh(y)
439         q=1-e
440         q=q[q>0.2]
441         q=q[:N]
442 
443         return q
444 
445     def
…   drawSourcePopulation(self,number,sourceplaneoverdensity=10,returnmasses=
…   False):
446         source_index=numpy.random.randint(0,len(self.zc),number*3)
447         #source_index=source_index[((self.zc[source_index]<10) &
```

```
447… (self.zc[source_index]>0.05))]
448         source_index=source_index[:number]
449         self.zs=self.zc[source_index]
450         self.Mvs=self.Mv[source_index]
451         self.ms={}
452         for band in self.bands:
453             if band !="VIS":
454                 self.ms[band]=self.m[band][source_index]
455             else:
456 
…   self.ms[band]=(self.m["r_SDSS"][source_index]+self.m["i_SDSS"][
…   source_index]+self.m["z_SDSS"][source_index])/3.
457 
458         self.r_phys=self.RofMz(self.Mvs,self.zs,scatter=True)
459         self.rs=self.draw_apparent_size(self.r_phys,self.zs)
460         self.qs=self.draw_flattening(number)
461 
462         self.ps=numpy.random.random_sample(number )*180
463 
464         #cosmos has a source density of ~0.015 per square arcsecond
465         if self.population=="cosmos":
466             fac=(0.015)**-0.5
467             a=fac*(sourceplaneoverdensity)**-.5
468         #lsst sim has a source density of ~0.06 per square arcsecond
469         elif self.population=="lsst":
470             fac=(0.06)**-0.5
471             a=fac*(sourceplaneoverdensity)**-.5
472 
473         else:
474             pass
475 
476         self.xs=(numpy.random.random_sample(number)-0.5)*a
477         self.ys=(numpy.random.random_sample(number)-0.5)*a
478 
479         if returnmasses:
480             self.mstar_src=self.mstar[source_index]
481             self.mhalo_src=self.mhalo[source_index]
482             return
…   self.zs,self.ms,self.xs,self.ys,self.qs,self.ps,self.rs,self.mstar_src,
…   self.mhalo_src
483 
484         return self.zs,self.ms,self.xs,self.ys,self.qs,self.ps,self.rs
485 
486 
487 class AnalyticSourcePopulation_(SourcePopulation_):
488     def  __init__(self,D=None,reset=False,
489 
…   bands=['F814W_ACS','g_SDSS','r_SDSS','i_SDSS','z_SDSS','Y_UKIRT'],cosmo=
…   [0.3,0.7,0.7]
```

```
                    ):
        self.bands=bands
        self.beginRedshiftDependentRelation(D,reset)
        print "not written!"



if __name__=="__main__":
    #RedshiftDependentRelation(reset=True)

    #L=LensPopulation_(reset=True,sigfloor=100)

    S=SourcePopulation_(reset=False,population="cosmos")
    S2=SourcePopulation_(reset=False,population="lsst")


    print
numpy.median(S.Mv[S.m["i_SDSS"]<25])-numpy.median(S2.Mv[S2.m["i_SDSS"]<
25])
    print
len(S.Mv[S.m["i_SDSS"]<25])*1./(len(S2.Mv[S2.m["i_SDSS"]<25])*100)
    print len(S.Mv)/(60.**2)/2.
    print len(S2.Mv[S2.m["i_SDSS"]<25])/(0.2**2)/(60.**2)
    print len(S2.Mv)/(0.2**2)/(60.**2)

    #print EarlyTypeRelations(self,100,z=None,scatter=True,band=None)
```

```
"""
A module to compute cosmological distances, including:
    comoving_distance (Dc)
    angular_diameter_distance (Da)
    luminosity_distance (Dl)
    comoving_volume (volume)

"""
c = 299792458.
G = 4.3e-6
from math import pi
import warnings
warnings.warn("Default cosmology is Om=0.3,Ol=0.7,h=0.7,w=-1 and
distance units are Mpc!",ImportWarning)

class Distance():
    def __init__(self,cosmo=[0.3,0.7,0.7]):
        self.OMEGA_M = cosmo[0]
        self.OMEGA_L = cosmo[1]
        self.h = cosmo[2]
        self.w = -1.
        self.wpars = None
        self.w_analytic = False
        self.Dc = self.comoving_distance
        self.Dt = self.comoving_transverse_distance
        self.Dm = self.comoving_transverse_distance
        self.Da = self.angular_diameter_distance
        self.Dl = self.luminosity_distance
        self.dm = self.distance_modulus
        self.volume = self.comoving_volume

    def set(self,cosmo):
        self.OMEGA_M = cosmo[0]
        self.OMEGA_L = cosmo[1]
        self.h = cosmo[2]

    def reset(self):
        self.OMEGA_M = 0.3
        self.OMEGA_L = 0.7
        self.h = 0.7
        self.w = -1.

    def age(self,z):
        from scipy import integrate
        f = lambda zp,m,l,k : (m/zp+k+l*zp**2)**-0.5
        om = self.OMEGA_M
        ol = self.OMEGA_L
        ok = 1.-om-ol
        return
```

```
48… (9.778/self.h)*integrate.romberg(f,1e-300,1/(1.+z),(om,ol,ok))
49
50      def comoving_distance(self,z1,z2=0.):
51          from scipy import integrate
52          if z2<z1:
53              z1,z2 = z2,z1
54          def fa(z):
55              if self.w_analytic==True:
56                  return self.w(z,self.wpars)
57              from math import exp
58              wa = lambda z : (1.+self.w(z,self.wpars))/(1.+z)
59              #return exp(3.*integrate.romberg(wa,0,z))
60              return exp(3.*integrate.quad(wa,0,z)[0])
61          if type(self.w)==type(self.comoving_distance) or
… type(self.w)==type(fa):
62              f = lambda z,m,l,k : (m*(1.+z)**3+k*(1.+z)**2+l*fa(z))**-0.5
63          elif self.w!=-1.:
64              f = lambda z,m,l,k :
… (m*(1.+z)**3+k*(1.+z)**2+l*(1.+z)**(3.*(1.+self.w)))**-0.5
65          else:
66              f = lambda z,m,l,k : (m*(1.+z)**3+k*(1.+z)**2+l)**-0.5
67          om = self.OMEGA_M
68          ol = self.OMEGA_L
69          ok = 1.-om-ol
70  #         return (c/self.h)*integrate.romberg(f,z1,z2,(om,ol,ok))/1e5
71          return (c/self.h)*integrate.quad(f,z1,z2,(om,ol,ok))[0]/1e5
72
73      def comoving_transverse_distance(self,z1,z2=0.):
74          dc = 1e5*self.comoving_distance(z1,z2)/(c/self.h)
75          ok = 1.-self.OMEGA_M-self.OMEGA_L
76          if ok>0:
77              from math import sinh,sqrt
78              dtc = sinh(sqrt(ok)*dc)/sqrt(ok)
79          elif ok<0:
80              from math import sin,sqrt
81              ok *= -1.
82              dtc = sin(sqrt(ok)*dc)/sqrt(ok)
83          else:
84              dtc = dc
85          return (c/self.h)*dtc/1e5
86
87      def angular_diameter_distance(self,z1,z2=0.):
88          if z2<z1:
89              z1,z2 = z2,z1
90          return self.comoving_transverse_distance(z1,z2)/(1.+z2)
91
92      def luminosity_distance(self,z):
93          return (1.+z)*self.comoving_transverse_distance(z)
94
```

```
95      def comoving_volume(self,z1,z2=0.):
96          from scipy import integrate
97          if z2<z1:
98              z1,z2 = z2,z1
99          f = lambda z,m,l,k:
 …  (self.comoving_distance(0.,z)**2)/((m*(1.+z)**3+k*(1.+z)**2+l)**0.5)
100         om = self.OMEGA_M
101         ol = self.OMEGA_L
102         ok = 1.-om-ol
103         return 4*pi*(c/self.h)*integrate.romberg(f,z1,z2,(om,ol,ok))/1e5
104
105     def rho_crit(self,z):
106         H2 = (self.OMEGA_M*(1+z)**3 + self.OMEGA_L)*(self.h/10.)**2
107         return 3*H2/(8.*pi*G)
108
109     def distance_modulus(self,z):
110         from math import log10
111         if z>0:return 5*log10(self.luminosity_distance(z)*1e5)
112         else: return 0
113
```

```
from __init__ import *
import cPickle
#import pyfits
import sys
import pylab as plt
import time
sigfloor=200

L=LensSample(reset=False,sigfloor=sigfloor,cosmo=[0.3,0.7,0.7])

experiment="Euclid"
frac=0.1

a=20#SN threshold
b=3#Magnification threshold

c=1000
d=1000


#experiment="DES"
if len(sys.argv)>1:
    experiment=sys.argv[1]
    frac=float(sys.argv[2])
if len(sys.argv)>3:
    a=int(sys.argv[3])
    b=int(sys.argv[4])
    #c=int(sys.argv[5])
    #d=int(sys.argv[6])

firstod=1
nsources=1


surveys=[]

if experiment=="Euclid":
    surveys+=["Euclid"]
if experiment=="CFHT":
    surveys+=["CFHT"] #full coadd (Gaussianised)
if experiment=="CFHTa":
    surveys+=["CFHTa"] #dummy CFHT
if experiment=="DES":
    surveys+=["DESc"] #Optimal stacking of data
    surveys+=["DESb"] #Best Single epoch image
    surveys+=["DESa"] #full coadd (Gaussianised)
if experiment=="LSST":
    surveys+=["LSSTc"] #Optimal stacking of data
    surveys+=["LSSTb"] #Best Single epoch image
```

```
    surveys+=["LSSTa"] #full coadd (Gaussianised)
    #print "only doing LSSTc"


S={}
n={}
for survey in surveys:
    S[survey]=FastLensSim(survey,fractionofseeing=1)
    S[survey].bfac=float(2)
    S[survey].rfac=float(2)


t0=time.clock()

#for sourcepop in ["lsst","cosmos"]:
for sourcepop in ["lsst"]:
  chunk=0
  Si=0
  SSPL={}
  foundcount={}
  for survey in surveys:
      foundcount[survey]=0

  if sourcepop=="cosmos":
      nall=1100000
  elif sourcepop=="lsst":
      nall=12530000
  nall=int(nall*frac)

  for i in range(nall):
    if i%10000==0:
        print "about to load"
        L.LoadLensPop(i,sourcepop)
        print i,nall

    if i!=0:
        if i%10000==0 or i==100 or i==300 or i==1000 or i==3000:
            t1=time.clock()
            ti=(t1-t0)/float(i)
            tl=(nall-i)*ti
            tl/=60#mins
            hl=numpy.floor(tl/(60))
            ml=tl-(hl*60)
            print i,"%ih%im left"%(hl,ml)

    lenspars=L.lens[i]
    if lenspars["lens?"]==False:
        del L.lens[i]
        continue
```

```
99
100     lenspars["rl"]["VIS"]=(lenspars["rl"]["r_SDSS"]+\
101
…   lenspars["rl"]["i_SDSS"]+lenspars["rl"]["z_SDSS"])/3
102     for mi in [lenspars["ml"],lenspars["ms"][1]]:
103         mi["VIS"]=(mi["r_SDSS"]+mi["i_SDSS"]+mi["z_SDSS"])/3
104
105
106
107
108     #if lenspars["zl"]>1 or lenspars["zl"]<0.2 or
…   lenspars["ml"]["i_SDSS"]<17 or lenspars["ml"]["i_SDSS"]>22:continue#
…   this is a CFHT compare quick n dirty test
109
110     lenspars["mag"]={}
111     lenspars["msrc"]={}
112     lenspars["mag"]={}
113     lenspars["msrc"]={}
114     lenspars["SN"]={}
115     lenspars["bestband"]={}
116     lenspars["pf"]={}
117     lenspars["resolved"]={}
118     lenspars["poptag"]={}
119     lenspars["seeing"]={}
120     lenspars["rfpf"]={}
121     lenspars["rfsn"]={}
122
123     lastsurvey="non"
124     for survey in surveys:
125
126
…   S[survey].setLensPars(lenspars["ml"],lenspars["rl"],lenspars["ql"],reset
…   =True)
127         for j in range(nsources):
128
…   S[survey].setSourcePars(lenspars["b"][j+1],lenspars["ms"][j+1],\
129
…   lenspars["xs"][j+1],lenspars["ys"][j+1],\
130
…   lenspars["qs"][j+1],lenspars["ps"][j+1],\
131                                     lenspars["rs"][j+1],sourcenumber=j+1
…   )
132
133         if survey[:3]+str(i)!=lastsurvey:
134             model=S[survey].makeLens(stochasticmode="MP")
135             SOdraw=numpy.array(S[survey].SOdraw)
136             if type(model)!=type(None):
137                 lastsurvey=survey[:3]+str(i)
138             if S[survey].seeingtest=="Fail":
```

```
                lenspars["pf"][survey]={}
                lenspars["rfpf"][survey]={}
                for src in S[survey].sourcenumbers:
                    lenspars["pf"][survey][src]=False
                    lenspars["rfpf"][survey][src]=False
                continue#try next survey
        else:
            S[survey].loadModel(model)
            S[survey].stochasticObserving(mode="MP",SOdraw=SOdraw)
            if S[survey].seeingtest=="Fail":
                lenspars["pf"][survey]={}
                for src in S[survey].sourcenumbers:
                    lenspars["pf"][survey][src]=False
                continue#try next survey
            S[survey].ObserveLens()


mag,msrc,SN,bestband,pf=S[survey].SourceMetaData(SNcutA=a,magcut=b,
SNcutB=[c,d])
        lenspars["SN"][survey]={}
        lenspars["bestband"][survey]={}
        lenspars["pf"][survey]={}
        lenspars["resolved"][survey]={}
        lenspars["poptag"][survey]=i
        lenspars["seeing"][survey]=S[survey].seeing
        rfpf={}
        rfsn={}
        for src in S[survey].sourcenumbers:
            rfpf[src]=False
            rfsn[src]=[0]
            lenspars["mag"][src]=mag[src]
            lenspars["msrc"][src]=msrc[src]
            lenspars["SN"][survey][src]=SN[src]
            lenspars["bestband"][survey][src]=bestband[src]
            lenspars["pf"][survey][src]=pf[src]
            lenspars["resolved"][survey][src]=S[survey].resolved[src]
        if survey!="Euclid":
            if S[survey].seeingtest!="Fail":
                if survey not in ["CFHT","CFHTa"]:

S[survey].makeLens(noisy=True,stochasticmode="1P",SOdraw=SOdraw,
MakeModel=False)

rfpf,rfsn=S[survey].RingFinderSN(SNcutA=a,magcut=b,SNcutB=[c,d],mode="
donotcrossconvolve")
                else:

rfpf,rfsn=S[survey].RingFinderSN(SNcutA=a,magcut=b,SNcutB=[c,d],mode="
crossconvolve")
```

```
180         lenspars["rfpf"][survey]=rfpf
181         lenspars["rfsn"][survey]=rfsn
182 
183         ###
184         #This is where you can add your own lens finder
185         #e.g.
186         #found=Myfinder(S[survey].image,S[survey].sigma,\
187         #                    S[survey].psf,S[survey].psfFFT)
188         #NB/ image,sigma, psf, psfFFT are dictionaries
189         #    The keywords are the filters, e.g. "g_SDSS", "VIS" etc
190 
191         #then save any outputs you'll need to the lenspars dictionary:
192         #lenspars["my_finder_result"]=found
193 
194         ###
195 
196         #If you want to save the images (it may well be a lot of data!):
197         #import pyfits #(or the astropy equivalent)
198 
199         #folder="where_to_save_fits_images"
200         #folder="%s/%i"%(folder,i)
201         #for band in S[survey].bands:
202             #img=S[survey].image[band]
203             #sig=S[survey].sigma[band]
204             #psf=S[survey].psf[band]
205             #resid=S[survey].fakeResidual[0][band]#The lens subtracted
206 
207         #resid contains the lensed source, with the lens subtracted
208         #assuming the subtraction is poisson noise limited (i.e. ideal)
209 
210 
… #pyfits.PrimaryHDU(img).writeto("%s/image_%s.fits"%(folder,band),\
211                 #                               clobber=True)
212 
… #pyfits.PrimaryHDU(sig).writeto("%s/sigma_%s.fits"%(folder,band),\
213                 #                               clobber=True)
214 
… #pyfits.PrimaryHDU(psf).writeto("%s/psf_%s.fits"%(folder,band),\
215                 #                               clobber=True)
216 
… #pyfits.PrimaryHDU(resid).writeto("%s/galsub_%s.fits"%(folder,band),
… clobber=True)
217 
218         ###
219 
220         L.lens[i]=None #delete used data for memory saving
221 
222     accept=False
223     for survey in surveys:
```

```
224             if lenspars["pf"][survey][1]:
225                 accept=True
226 
227         if accept:
228             #S[survey].display(band="VIS",bands=["VIS","VIS","VIS"])
229             #if Si>100:exit()
230             Si+=1
231             SSPL[Si]=lenspars.copy()
232             if (Si+1)%1000==0:
233 
…   f=open("LensStats/%s_%s_Lens_stats_%i.pkl"%(experiment,sourcepop,chunk),
…   "wb")
234                 cPickle.dump([frac,SSPL],f,2)
235                 f.close()
236                 SSPL={} # reset SSPL or memory fills up
237                 chunk+=1
238 
239         del L.lens[i]
240 
241 
…   f=open("LensStats/%s_%s_Lens_stats_%i.pkl"%(experiment,sourcepop,chunk),
…   "wb")
242       cPickle.dump([frac,SSPL],f,2)
243       f.close()
244       print Si
245 
246 bl=[]
247 for j in SSPL.keys():
248         try:
249             if SSPL[j]["rfpf"][survey][1]:
250                 bl.append(SSPL[j]["b"][1])
251         except KeyError:pass
252 
```

```
import indexTricks as iT
import numpy
from pylens import *
from imageSim import profiles,convolve,SBModels
import distances as D
import cPickle
import indexTricks as iT
import numpy,pylab
from imageSim import profiles,convolve,models
import pylab as plt
from Surveys import Survey
from StochasticObserving import SO
from SignaltoNoise import S2N

class FastLensSim(SO,S2N):
    def __init__(self,surveyname,fractionofseeing=1):
        #--------------------------------------------------
        ### Read in survey
        self.surveyname=surveyname
        survey=Survey(surveyname)#This stores typical survey  in
…Surveys.Survey
        self.survey=survey
        self.pixelsize=survey.pixelsize
        self.side=survey.side
        self.readnoise=survey.readnoise
        self.nexposures=survey.nexposures
        self.f_sky=survey.f_sky

        self.bands=survey.bands
        self.strategy=survey.strategy
        self.strategyx=survey.strategyx

        self.exposuretimes={}
        self.zeropoints={}
        self.stochasticobservingdata={}
        self.gains={}
        self.seeing={}
        self.psfscale={}
        self.psf={}
        self.psfFFT={}

        self.ET={}
        self.SB={}


        for i in range(len(survey.bands)):
            self.exposuretimes[survey.bands[i]]=survey.exposuretimes[i]
            self.zeropoints[survey.bands[i]]=survey.zeropoints[i]
            self.gains[survey.bands[i]]=survey.gains[i]
```

```
49
… self.stochasticobservingdata[survey.bands[i]]=survey.
… stochasticobservingdata[i]
50          self.zeroexposuretime=survey.zeroexposuretime
51          #--------------------------------------------------
52          ###do some setup
53      self.xl=(self.side-1.)/2.
54      self.yl=(self.side-1.)/2.
55      self.x,self.y = iT.coords((self.side,self.side))
56          self.r2 = (self.x-self.xl)**2+(self.y-self.yl)**2
57
58          self.pixelunits=False
59
60          #--------------------------------------------------
61          self.Reset()
62
63      def Reset(self):
64          self.sourcenumbers=[]
65          #Some objects that need pre-defining as dictionaries
66          self.magnification={}
67          self.image={}
68          self.sigma={}
69          self.residual={}
70          self.zeromagcounts={}
71          self.xs={}
72          self.ys={}
73          self.ms={}
74          self.qs={}
75          self.ps={}
76          self.rs={}
77          self.ns={}
78          self.bl={}
79          self.src={}
80          self.galmodel={}
81          self.sourcemodel={}
82          self.model={}
83          self.totallensedsrcmag={}
84          self.fakeLens={}
85          self.image={}
86          self.sigma={}
87          self.fakeResidual={}
88          self.fakeResidual[0]={}
89          self.SN={}
90          self.SNRF={}
91          self.convolvedsrc={}
92          self.convolvedgal={}
93
94  #=============================================================================
… ====
```

```
95
96      def trytoconvert(self,par,p):
97          try:return par/p
98          except NameError:print "warning one of the parameters is not
…   defined"
99
100 #=========================================================================
…   ====
101
102     def
…   setLensPars(self,m,r,q,n=4,pixelunits=False,reset=True,xb=0,xp=0,jiggle=
…   0):
103         if reset: self.Reset()
104         self.rl={}
105         if pixelunits==False:
106             for band in r.keys():
107                 self.rl[band]=self.trytoconvert(r[band],self.pixelsize)
108         self.ml=m
109         self.ql=q
110
111         self.deltaxl=(numpy.random.rand()-0.5)*2*jiggle
112         self.deltayl=(numpy.random.rand()-0.5)*2*jiggle
113         if jiggle!=0:
114             self.deltap=0.+(numpy.random.rand()-0.5)*180
115             n=(numpy.random.rand()+1)*4
116         else:
117             self.deltap=0.
118
119         self.nl=n
120
…   self.gal=SBModels.Sersic('gal',{'x':self.xl+self.deltaxl,'y':self.yl+
…   self.deltayl,'q':self.ql,'pa':90+self.deltap,'re':self.rl[band],'n':self
…   .nl})
121
122         self.xb=xb
123         self.xp=xp
124
125
126 #=========================================================================
…   ====
127
128     def
…   setSourcePars(self,b,m,x,y,q,p,r,n=1,pixelunits=False,sourcenumber=1):
129         if pixelunits==False:
130             x=self.trytoconvert(x,self.pixelsize)
131             y=self.trytoconvert(y,self.pixelsize)
132             r=self.trytoconvert(r,self.pixelsize)
133             b=self.trytoconvert(b,self.pixelsize)
134         self.xs[sourcenumber]=x+self.xl+self.deltaxl+0.000001
```

```
135         self.ys[sourcenumber]=y+self.yl+self.deltayl+0.000001
136         self.ms[sourcenumber]=m
137         self.qs[sourcenumber]=q
138         self.ps[sourcenumber]=p
139         self.rs[sourcenumber]=r
140         self.ns[sourcenumber]=n
141         self.bl[sourcenumber]=b
142         self.src[sourcenumber]=SBModels.Sersic('src%i'%sourcenumber,
143             {'x':self.xs[sourcenumber],'y':self.ys[sourcenumber],\
144              'q':self.qs[sourcenumber],'pa':self.ps[sourcenumber],\
145              're':self.rs[sourcenumber],'n':self.ns[sourcenumber]})
146         self.sourcenumbers.append(sourcenumber)
147         self.sourcemodel[sourcenumber]={}
148         self.totallensedsrcmag[sourcenumber]={}
149         self.fakeResidual[sourcenumber]={}
150         self.SN[sourcenumber]={}
151         self.SNRF[sourcenumber]={}
152         self.convolvedsrc[sourcenumber]={}
153
154 #=========================================================================
…   ====
155
156     def lensASource(self,sourcenumber,bands):
157         src=self.src[sourcenumber]
158
…   lens=massmodel.PowerLaw('lens',{},{'x':self.xl+self.deltaxl,'y':self.yl+
…   self.deltayl,'q':self.ql,'pa':90+self.deltap,'b':self.bl[sourcenumber],'
…   eta':1})
159
…   es=massmodel.ExtShear('lens',{},{'x':self.xl+self.deltaxl,'y':self.yl+
…   self.deltayl,'pa':self.xp,'b':self.xb})
160         lenses=[lens,es]
161
162         a=51
163         ox,oy=iT.coords((a,a))
164         ps=(self.rs[sourcenumber]*(10./a))
165         ox=(ox-(a-1)/2.)*ps+(self.xs[sourcenumber])
166         oy=(oy-(a-1)/2.)*ps+(self.ys[sourcenumber])
167
168         unlensedsrcmodel=(src.pixeval(ox,oy,csub=5)*(ps**2)).sum()
169         srcnorm=unlensedsrcmodel.sum()
170         unlensedsrcmodel/=srcnorm
171
172
…   srcmodel=pylens.lens_images(lenses,src,[self.x,self.y],getPix=True,csub=
…   5)[0]
173         srcmodel[srcmodel<0]=0
174         srcmodel/=srcnorm
175
```

```
176
…  self.magnification[sourcenumber]=(numpy.sum(numpy.ravel(srcmodel))/numpy
…  .sum(numpy.ravel(unlensedsrcmodel)))
177         sm={}
178         for band in bands:
179
…  unlensedtotalsrcflux=10**(-(self.ms[sourcenumber][band]-self.zeropoints[
…  band])/2.5)
180             sm[band]=srcmodel*unlensedtotalsrcflux
181
182             if sm[band].max()>0:
183
…  self.totallensedsrcmag[sourcenumber][band]=-2.5*numpy.log10(sm[band].sum
…  ())+self.zeropoints[band]
184             else:
185                 self.totallensedsrcmag[sourcenumber][band]=99
186         return sm
187
188 #=========================================================================
…  ====
189
190     def EvaluateGalaxy(self,light,mag,bands):
191         model={}
192         lightm=light.pixeval(self.x,self.y,csub=5)
193         lightm[lightm<0]=0
194         lightm/=lightm.sum()
195         for band in bands:
196             flux=10**(-(mag[band]-self.zeropoints[band])/2.5)
197             model[band]=lightm*flux
198
199         return model
200
201 #=========================================================================
…  ====
202
203     def MakeModel(self, bands):
204         #did you know that self.gal is actually fixed for all bands
…  currently?
205         self.galmodel=self.EvaluateGalaxy(self.gal,self.ml,bands)
206         for sourcenumber in self.sourcenumbers:
207
…  self.sourcemodel[sourcenumber]=self.lensASource(sourcenumber,bands)
208
209         for band in bands:
210             self.model[band]=self.galmodel[band]*1
211             for sourcenumber in self.sourcenumbers:
212                 self.model[band]+=self.sourcemodel[sourcenumber][band]
213
214 #=========================================================================
```

```
214… ====

215
216      def ObserveLens(self,noisy=True,bands=[]):
217        if bands==[]:bands=self.bands
218        for band in bands:
219          if self.seeing[band]!=0:
220            convolvedgal,self.psfFFT[band] =
…  convolve.convolve(self.galmodel[band],self.psf[band],True)
221            convolvedgal[convolvedgal<0]=0
222            self.convolvedgal[band]=convolvedgal
223
224
225            convolvedmodel=convolvedgal*1
226
227            convolvedsrc={}
228
229            for sourcenumber in self.sourcenumbers:
230
…  convolvedsrc[sourcenumber]=convolve.convolve(self.sourcemodel[
…  sourcenumber][band],self.psfFFT[band],False)[0]
231              convolvedsrc[sourcenumber][convolvedsrc[sourcenumber]<0]=0
232
…  self.convolvedsrc[sourcenumber][band]=convolvedsrc[sourcenumber]
233              convolvedmodel+=convolvedsrc[sourcenumber]
234
235
…  self.zeromagcounts[band]=(10**(-(0-self.zeropoints[band])/2.5))
236
237
…  exposurecorrection=((self.ET[band]*1./self.zeroexposuretime))*self.gains
…  [band]
238            convolvedmodel*=exposurecorrection
239
240            #skybackground per second per square arcsecond
241
…  background=(10**(-(self.SB[band]-self.zeropoints[band])/2.5))*(self.
…  pixelsize**2)
242            tot_bg=background*exposurecorrection
243
244
245
…  sigma=((convolvedmodel+tot_bg)+self.nexposures*(self.readnoise**0.5)**2)
…  **.5
246
247            fakeLens=convolvedmodel*1.
248            if
…  noisy:fakeLens+=(numpy.random.randn(self.side,self.side)*(sigma))
249
250            #convert back to ADU/second:
```

```
251             fakeLens/=exposurecorrection
252             sigma/=exposurecorrection
253 
254             self.image[band]=fakeLens*1
255             self.fakeLens[band]=fakeLens*1
256             self.sigma[band]=sigma*1
257             self.fakeResidual[0][band]=fakeLens-convolvedgal
258             for sourcenumber in self.sourcenumbers:
259                 self.SN[sourcenumber][band]=self.SNfunc(\
260                     convolvedsrc[sourcenumber],sigma)
261                 self.fakeResidual[sourcenumber][band]=\
262                     fakeLens-convolvedmodel+convolvedsrc[sourcenumber]
263 
264 #=========================================================================
    ====
265     def loadModel(self,ideallens):
266 
    self.galmodel,self.sourcemodel,self.model,self.magnification,self.
    totallensedsrcmag=ideallens
267         self.image=self.model
268 
269 #=========================================================================
    ====
270 
271     def loadConvolvedModel(self,ideallens):
272 
    self.galmodel,self.sourcemodel,self.model,self.magnification,self.
    totallensedsrcmag=ideallens
273         self.image=self.model
274 
275 
276 #=========================================================================
    ====
277 
278     def
    makeLens(self,stochastic=True,save=False,noisy=True,stochasticmode="MP",
    SOdraw=[],bands=[],musthaveallbands=False,MakeModel=True):
279         if
    stochastic==True:self.stochasticObserving(mode=stochasticmode,SOdraw=
    SOdraw,musthaveallbands=musthaveallbands)
280         if self.seeingtest=="Fail":return None
281         if bands==[]:bands=self.bands
282 
283         if MakeModel:
284             self.MakeModel(bands)
285 
286         if self.strategy=="resolve":
287             if
    stochastic==True:self.stochasticObserving(mode=stochasticmode,SOdraw=
```

```
287…  SOdraw) #have to rerun stochastic observing now we know the
  …  magnification
288
289           self.ObserveLens(noisy=noisy)
290           return
  …  [self.galmodel,self.sourcemodel,self.model,self.magnification,self.
  …  totallensedsrcmag]
291
292
293  #=============================================================================
  …  ====
294
295
296       def
  …  makeColorLens(self,bands=["g_SDSS","r_SDSS","i_SDSS"],recolourize=True):
297           if self.surveyname=="Euclid" and
  …  bands==["g_SDSS","r_SDSS","i_SDSS"]:
298               bands=["VIS","VIS","VIS"]
299           import colorImage
300           goodbands=[]
301           for band in bands:
302               try:
303                   self.image[band]
304                   goodbands.append(band)
305               except KeyError:
306                   pass
307           bands=goodbands
308           if len(bands)==1:
309               bands=[bands[0],bands[0],bands[0]]
310           if len(bands)==2:
311               bands=[bands[0],"dummy",bands[1]]
312               self.ml["dummy"]=(self.ml[bands[0]]+self.ml[bands[2]])/2
313
  …  self.image["dummy"]=(self.image[bands[0]]+self.image[bands[2]])/2
314           if recolourize:
315               self.color = colorImage.ColorImage()
316               self.color.bMinusr=(self.ml[bands[0]]-self.ml[bands[2]])/4.
317               self.color.bMinusg=(self.ml[bands[0]]-self.ml[bands[1]])/4.
318               self.color.nonlin=4.
319               self.colorimage = self.color.createModel(\
320
  …  self.image[bands[0]],self.image[bands[1]],self.image[bands[2]])
321           else:
322               self.colorimage = self.color.colorize(\
323
  …  self.image[bands[0]],self.image[bands[1]],self.image[bands[2]])
324
325           return self.colorimage
326
```

```
327
328 #============================================================================
    ====
329
330     def display(self,band="g_SDSS",bands=["g_SDSS","r_SDSS","i_SDSS"]):
331         if self.surveyname=="Euclid":bands=["VIS","VIS","VIS"]
332         import pylab as plt
333         plt.ion()
334         plt.figure(1)
335         plt.imshow(self.makeColorLens(bands=bands),interpolation="none")
336         plt.figure(2)
337         import colorImage
338         self.color = colorImage.ColorImage()#sigma-clipped single band
    residual
339
    plt.imshow(self.color.createModel(self.fakeResidual[0][band],self.
    fakeResidual[0][band],self.fakeResidual[0][band])[:,:,0],interpolation="
    none")
340         plt.figure(3)
341         plt.imshow(self.fakeResidual[0][band],interpolation="none")
342         try:
343             self.fakeResidual[1]["RF"]
344             plt.figure(4)
345             plt.imshow(self.fakeResidual[1]["RF"],interpolation="none")
346         except KeyError: pass
347
348         plt.draw()
349         raw_input()
350         plt.ioff()
351
352 #============================================================================
    ====
353
354     def
    Rank(self,mode,band="g_SDSS",bands=["g_SDSS","r_SDSS","i_SDSS"]):
355         import pylab as plt
356         plt.ion()
357         rank="d"
358         while rank not in ["0","1","2","3","4","-1","-2","-3"]:
359             if mode=="colour":
360
    plt.imshow(self.makeColorLens(bands=bands),interpolation="none")
361                 plt.draw()
362             if mode=="rf":
363
    plt.imshow(self.fakeResidual[0]["RF"],interpolation="none")
364                 plt.draw()
365             if mode=="best":
366
```

```
366… plt.imshow(self.fakeResidual[0][band],interpolation="none")
367                 plt.draw()
368             rank=raw_input()
369             if rank=="":rank="0"
370         plt.ioff()
371         return rank
372 #=========================================================================
…   ====
373
```

```python
import SBProfiles
from pointSource import PixelizedModel as PM, GaussianModel as GM
from math import pi

def cnts2mag(cnts,zp):
    from math import log10
    return -2.5*log10(cnts) + zp

_SersicPars = [['amp','n','pa','q','re','x','y'],
               ['logamp','n','pa','q','re','x','y'],
               ['amp','n','q','re','theta','x','y'],
               ['logamp','n','q','re','theta','x','y']]

class SBModel:
    def __init__(self,name,pars,convolve=0):
        if 'amp' not in pars.keys():
            pars['amp'] = 1.
        self.keys = pars.keys()
        self.keys.sort()
        if self.keys not in self._SBkeys:
            import sys
            print 'Not all (or too many) parameters were defined!'
            sys.exit()
        self._baseProfile.__init__(self)
        self.vmap = {}
        self.pars = pars
        for key in self.keys:
            try:
                v = self.pars[key].value
                self.vmap[key] = self.pars[key]
            except:
                self.__setattr__(key,self.pars[key])
        self.setPars()
        self.name = name
        self.convolve = convolve


    def __setattr__(self,key,value):
        if key=='pa':
            self.__dict__['pa'] = value
            if value is not None:
                self.__dict__['theta'] = value*pi/180.
        elif key=='theta':
            if value is not None:
                self.__dict__['pa'] = value*180./pi
            self.__dict__['theta'] = value
        elif key=='logamp':
            if value is not None:
                self.__dict__['amp'] = 10**value
```

```
50          else:
51              self.__dict__[key] = value
52
53
54      def setPars(self):
55          for key in self.vmap:
56              self.__setattr__(key,self.vmap[key].value)
57
58
59  class Sersic(SBModel,SBProfiles.Sersic):
60      _baseProfile = SBProfiles.Sersic
61      _SBkeys = [['amp','n','pa','q','re','x','y'],
62                 ['logamp','n','pa','q','re','x','y'],
63                 ['amp','n','q','re','theta','x','y'],
64                 ['logamp','n','q','re','theta','x','y']]
65
66      def __init__(self,name,pars,convolve=0):
67          SBModel.__init__(self,name,pars,convolve)
68
69      def getMag(self,amp,zp):
70          from scipy.special import gamma
71          from math import exp,pi
72          n = self.n
73          re = self.re
74          k = 2.*n-1./3+4./(405.*n)+46/(25515.*n**2)
75          cnts = (re**2)*amp*exp(k)*n*(k**(-2*n))*gamma(2*n)*2*pi
76          return cnts2mag(cnts,zp)
77
78      def Mag(self,zp):
79          return self.getMag(self.amp,zp)
80
81
82  class Gauss(SBModel,SBProfiles.Gauss):
83      _baseProfile = SBProfiles.Gauss
84      _SBkeys = [['amp','pa','q','r0','sigma','x','y']]
85
86      def __init__(self,name,pars,convolve=0):
87          if 'r0' not in pars.keys():
88              pars['r0'] = None
89          SBModel.__init__(self,name,pars,convolve)
90
91      def getMag(self,amp,zp):
92          from math import exp,pi
93          if self.r0 is None:
94              cnts = amp/(2*pi*self.sigma**2)
95          else:
96              from scipy.special import erf
97              r0 = self.r0
98              s = self.sigma
```

```
            r2pi = (2*pi)**0.5
            cnts =
amp*pi*s*(r2pi*r0*(1.+erf(r0/(s*2**0.5)))+2*s*exp(-0.5*r0**2/s**2))
        return cnts2mag(cnts,zp)

    def Mag(self,zp):
        return self.getMag(self.amp,zp)



class PointSource(GM,PM):
    def __init__(self,name,model,var=None,const=None):
        if const is None:
            const = {}
        if var is None:
            var = {}
        keys = var.keys()+const.keys()
        keys.sort()
        if keys!=['amp','x','y']:
            print "Not all parameters defined!",keys
            df
        self.keys = keys
        self.values = {}
        self.vmap = {}
        self.ispix = False
        for key in var.keys():
            self.values[key] = None
            self.vmap[var[key]] = key
        for key in const.keys():
            self.values[key] = const[key]
        if type(model)==type([]):
            GM.__init__(self,model)
        else:
            PM.__init__(self,model)
            self.ispix = True
        self.setValues()
        self.name = name
        self.convolve = None

    def __setattr__(self,key,value):
        if key=='logamp':
            if value is not None:
                self.__dict__['amp'] = 10**value
        else:
            self.__dict__[key] = value

    def pixeval(self,xc,yc,dummy1=None,dummy2=None,**kwargs):
        if self.ispix==True:
            return PM.pixeval(self,xc,yc)
```

```
147         else:
148             return GM.pixeval(self,xc,yc)
149 
150     def setValues(self):
151         self.x = self.values['x']
152         self.y = self.values['y']
153         if 'amp' in self.keys:
154             self.amp = self.values['amp']
155         elif self.values['logamp'] is not None:
156             self.amp = 10**self.values['logamp']
157 
158     def getMag(self,amp,zp):
159         return cnts2mag(amp,zp)
160 
161     def Mag(self,zp):
162         return self.getMag(self.amp,zp)
163 
164     def setPars(self,pars):
165         for key in self.vmap:
166             self.values[self.vmap[key]] = pars[key]
167         self.setValues()
168 
```

```
import numpy,time
from scipy import interpolate
"""
MINIMAL ERROR CHECKING!
"""
def cnts2mag(cnts,zp):
    from math import log10
    return -2.5*log10(cnts) + zp


class Sersic:
    def
__init__(self,x=None,y=None,q=None,pa=None,re=None,amp=None,n=None):
        self.x = x
        self.y = y
        self.q = q
        self.pa = pa
        self.re = re
        self.amp = amp
        self.n = n
        self.convolve = True

        #A flag to tell the code not to pixeval every step, if nothing
changes
        self.NoFreeParams=False

    def setAmpFromMag(self,mag,zp):
        from math import exp,log10,pi
        from scipy.special import gamma
        cnts = 10**(-0.4*(mag-zp))
        n = self.n
        re = self.re
        k = 2.*n-1./3+4./(405.*n)+46/(25515.*n**2)
        self.amp = cnts/((re**2)*exp(k)*n*(k**(-2*n))*gamma(2*n)*2*pi)

    def eval(self,r):
        k = 2.*self.n-1./3+4./(405.*self.n)+46/(25515.*self.n**2)
        R = r/self.re
        return self.amp*numpy.exp(-k*(R**(1./self.n) - 1.))

    def pixeval(self,x,y,scale=1,csub=23):
        if self.NoFreeParams==True:
            try:
                return self.pixevalresult
            except:
                pass

        from scipy import interpolate
        from math import pi,cos as COS,sin as SIN
```

```
48          shape = x.shape
49          x = x.ravel()
50          y = y.ravel()
51
52          cos = COS(self.pa*pi/180.)
53          sin = SIN(self.pa*pi/180.)
54          xp = (x-self.x)*cos+(y-self.y)*sin
55          yp = (y-self.y)*cos-(x-self.x)*sin
56          r = (self.q*xp**2+yp**2/self.q)**0.5
57
58          k = 2.*self.n-1./3+4./(405.*self.n)+46/(25515.*self.n**2)
59          R = numpy.logspace(-5.,4.,451) # 50 pnts / decade
60          s0 = numpy.exp(-k*(R**(1./self.n) - 1.))
61
62          # Determine corrections for curvature
63          rpow = R**(1./self.n - 1.)
64          term1 = (k*rpow/self.n)**2
65          term2 = k*(self.n-1.)*rpow/(R*self.n**2)
66          wid = scale/self.re
67          corr = (term1+term2)*wid**3/6.
68          try:
69              minR = R[abs(corr)<0.005].min()
70          except:
71              minR = 0
72
73          # Evaluate model!
74          model = interpolate.splrep(R,s0,k=3,s=0)
75          model2 = interpolate.splrep(R,s0*R*self.re**2,k=3,s=0)
76          R0 = r/self.re
77          s = interpolate.splev(R0,model)*scale**2
78          if self.n<=1. or minR==0:
79              return self.amp*s.reshape(shape)
80          coords = numpy.where(R0<minR)[0]
81          c =
… (numpy.indices((csub,csub)).astype(numpy.float32)-csub/2)*scale/csub
82          for i in coords:
83              # The central pixels are tricky because we can't assume that
… we
84              #   are integrating in delta-theta segments of an annulus;
… these
85              #   pixels are treated separately by sub-sampling with ~500
… pixels
86              if R0[i]<3*scale/self.re:
87                  s[i] = 0.
88                  y0 = c[1]+y[i]
89                  x0 = c[0]+x[i]
90                  xp = (x0-self.x)*cos+(y0-self.y)*sin
91                  yp = (y0-self.y)*cos-(x0-self.x)*sin
92                  r0 = (self.q*xp**2+yp**2/self.q)**0.5/self.re
```

```
93                  s[i] =
…  interpolate.splev(r0.ravel(),model).mean()*scale**2
94                  continue
95              lo = R0[i]-0.5*scale/self.re
96              hi = R0[i]+0.5*scale/self.re
97              angle = (scale/self.re)/R0[i]
98              s[i] = angle*interpolate.splint(lo,hi,model2)
99              # The following code should no longer be needed
100             """
101             if lo<0:
102                 s[i] =
…  ((interpolate.splint(0,abs(lo),model2)+interpolate.splint(0,hi,model2)))
…  *pi*2
103             else:
104                 s[i] = angle*interpolate.splint(lo,hi,model2)
105             """
106
107         self.pixevalresult=self.amp*s.reshape(shape)
108
109         return self.pixevalresult
110
111
112 class deV(Sersic):
113
114     def __init__(self,x=None,y=None,q=None,pa=None,re=None,amp=None):
115         Sersic.__init__(self,x,y,q,pa,re,amp,4.)
116
117
118 class exp(Sersic):
119
120     def __init__(self,x=None,y=None,q=None,pa=None,re=None,amp=None):
121         Sersic.__init__(self,x,y,q,pa,re,amp,1.)
122
123
124 class Gauss:
125
126     def
…  __init__(self,x=None,y=None,q=None,pa=None,sigma=None,amp=None,r0=None):
127         self.x = x
128         self.y = y
129         self.q = q
130         self.pa = pa
131         self.sigma = sigma
132         self.amp = amp
133         self.r0 = r0
134         self.convolve = True
135
136     def pixeval(self,x,y,factor=None,csub=None):
137         from math import pi
```

```
138
139         cos = numpy.cos(self.pa*pi/180.)
140         sin = numpy.sin(self.pa*pi/180.)
141         xp = (x-self.x)*cos+(y-self.y)*sin
142         yp = (y-self.y)*cos-(x-self.x)*sin
143         r2 = (self.q*xp**2+yp**2/self.q)
144         if self.r0 is None:
145             return self.amp*numpy.exp(-0.5*r2/self.sigma**2)
146         return
…   self.amp*numpy.exp(-0.5*(r2**0.5-self.r0)**2/self.sigma**2)
147
148
149     def getMag(self,zp):
150         from math import exp,pi
151         if self.r0 is None:
152             cnts = self.amp*(2*pi*self.sigma**2)
153         else:
154             from scipy.special import erf
155             r0 = self.r0
156             s = self.sigma
157             r2pi = (2*pi)**0.5
158             cnts =
…   self.amp*pi*s*(r2pi*r0*(1.+erf(r0/(s*2**0.5)))+2*s*exp(-0.5*r0**2/s**2))
159         return cnts2mag(cnts,zp)
160
161
162     def eval(self,x,y):
163         from math import pi
164         try:
165             cos = numpy.cos(self.theta)
166             sin = numpy.sin(self.theta)
167             xp = (x-self.x)*cos+(y-self.y)*sin
168             yp = (y-self.y)*cos-(x-self.x)*sin
169             r = (self.q*xp**2+yp**2/self.q)**0.5/self.sigma
170             s =
…   self.amp*numpy.exp(-0.5*r**self.n)/(2.*pi*self.sigma**2)**1.0
171             return s
172         except:
173             return x*0.
174
175
```

```
import cPickle,numpy
class Survey():
    def  __init__(self,Name):
        self.zeroexposuretime=1
        self.strategy="resolve"
        self.strategyx=1
        if Name[:3]=="DES":
            self.pixelsize=0.263
            self.side=76
            self.bands=['g','r','i']
            self.zeropoints=[30,30,30]
            self.zeroexposuretime=90.
            self.skybrightnesses=[21.7,20.7,20.1]
            self.exposuretimes=[900,900,900]
            self.gains=[4.5,4.5,4.5]
            self.seeing=[.9,.9,.9]
            self.nexposures=10
            self.degrees_of_survey=5000
            self.readnoise=(10/4.5)
            twodg=cPickle.load(open("2dpdfs/2dg_DES.pkl",'r'))
            twodr=cPickle.load(open("2dpdfs/2dr_DES.pkl",'r'))
            twodi=cPickle.load(open("2dpdfs/2di_DES.pkl",'r'))
            self.stochasticobservingdata=[twodg,twodr,twodi]
            if Name=="DESsv":
                self.degrees_of_survey=150
            if Name=="DESsv" or  Name =="DESa":
                self.strategy="absolute"
                self.strategyx=10
            if Name=="DESb":
                self.strategy="best"
                self.strategyx=1
            if Name=="DESdummy":
                self.strategy="absolute"
                self.strategyx=10
                dumg=numpy.array([[1.2,21.7],[1.2,21.7]])
                dumr=numpy.array([[0.95,20.7],[0.95,20.7]])
                dumi=numpy.array([[0.95,20.1],[0.95,20.1]])
                print "dummy seeing,strat"
                self.stochasticobservingdata=[dumg,dumr,dumi]
                self.strategy="absolute"
                self.strategyx=10



        elif Name[:4]=="LSST":
            self.pixelsize=0.18
            self.side=111
            self.bands=['g','r','i']
            self.zeropoints=[30,30,30]
```

```
50              self.zeroexposuretime=25
51              self.skybrightnesses=[21.7,20.7,20.1]
52              self.exposuretimes=[3000,6000,6000]
53              self.gains=[4.5,4.5,4.5]
54              self.seeing=[.4,.4,.4]
55              self.nexposures=100
56              self.degrees_of_survey=18000
57              self.readnoise=(10/4.5)
58              twodg=cPickle.load(open("2dpdfs/2dg_LSST.pkl",'r'))
59              twodr=cPickle.load(open("2dpdfs/2dr_LSST.pkl",'r'))
60              twodi=cPickle.load(open("2dpdfs/2di_LSST.pkl",'r'))
61              self.stochasticobservingdata=[twodg,twodr,twodi]
62              if Name[-1]=="a":
63                  self.strategy="absolute"
64                  self.strategyx=10
65              if Name[-1]=="b":
66                  self.strategy="best"
67                  self.strategyx=1
68              if Name[-1]=="c":
69                  self.strategy="resolve"
70                  self.strategyx=1
71
72
73
74          elif Name=="CFHT" or Name=="CFHTa":
75              self.pixelsize=0.187
76              self.side=107
77              self.bands=['g','r','i']
78              self.zeropoints=[26.96,26.47,26.24]
79              self.zeroexposuretime=1
80              self.skybrightnesses=[21.9,20.6,19.2]
81              self.exposuretimes=[3500,5500,5500]
82              self.gains=[1.62,1.62,1.62]
83              self.seeing=[.8,.8,0.8]
84              self.nexposures=1 # this isn't actually true, but the 2d
… pdfs
85                                  # are for the CFHT coadds (approximately)
86              self.degrees_of_survey=150
87              self.readnoise=(5)
88              twodg=cPickle.load(open("2dpdfs/2dg_CFHT.pkl",'r'))
89              twodr=cPickle.load(open("2dpdfs/2dr_CFHT.pkl",'r'))
90              twodi=cPickle.load(open("2dpdfs/2di_CFHT.pkl",'r'))
91              self.stochasticobservingdata=[twodg,twodr,twodi]
92              self.strategy="absolute"
93              self.strategyx=10
94
95          elif Name=="HSC":
96              self.pixelsize=0.17
97              self.side=200
```

```
98              self.bands=['g','r','i']
99              self.zeropoints=[30,30]
100             self.zeroexposuretime=90./(8.2/4)**2
101             self.skybrightnesses=[21.9,19.2]
102             self.exposuretimes=[600,600]
103             self.gains=[4.5,4.5]
104             self.seeing=[.8,.8]
105             self.nexposures=10
106             self.degrees_of_survey=1370
107             self.readnoise=(10/4.5)
108             twodg=numpy.array([[0.8,21.9],[0.8,21.9]])
109             twodi=numpy.array([[0.8,19.2],[0.8,19.2]])
110             self.stochasticobservingdata=[twodg,twodi]
111
112         elif Name=="COSMOS":
113             pass
114
115
116         elif Name=="Euclid":
117             self.pixelsize=0.1
118             self.side=200
119             self.bands=['VIS']
120             self.zeropoints=[25.5]
121             self.zeroexposuretime=1.
122             self.skybrightnesses=[22.2]
123             self.exposuretimes=[1610]
124             self.gains=[1]
125             self.seeing=[.2]
126             self.nexposures=4
127             self.degrees_of_survey=20000
128             self.readnoise=(4.5)
129             twodVIS=numpy.array([[0.17,22.2],[0.17,22.2]])
130             self.stochasticobservingdata=[twodVIS]
131
132         elif Name=="ideal":
133             self.pixelsize=0.05
134             self.side=400
135             self.bands=['g','r','i']
136             self.zeropoints=[24.5,24.5,24.5]
137             self.zeroexposuretime=4.
138             self.skybrightnesses=[220,220,220]
139             self.exposuretimes=[22400000000,22400000000,22400000000]
140             self.gains=[1,1,1]
141             self.seeing=[.05,0.05,0.05]
142             self.nexposures=1
143             self.readnoise=(.005)
144             twodr=numpy.array([[0.1,220],[0.1,220]])
145             twodg=numpy.array([[0.1,220],[0.1,220]])
146             twodi=numpy.array([[0.1,220],[0.1,220]])
```

```
147             self.stochasticobservingdata=[twodg,twodr,twodi]
148             self.degrees_of_survey=41253
149         else:
150             print "I don't know that survey"
151             exit()
152
153
154
155         #convert bandnames into the required formats
156         for i in range(len(self.bands)):
157             bandname=self.bands[i]
158             if bandname=="g":
159                 self.bands[i]="g_SDSS"
160             if bandname=="r":
161                 self.bands[i]="r_SDSS"
162             if bandname=="i":
163                 self.bands[i]="i_SDSS"
164             if bandname=="z":
165                 self.bands[i]="z_SDSS"
166             if bandname=="F814":
167                 self.bands[i]="F814W_ACS"
168
169         degrees_of_whole_sky=41253.
170         self.f_sky=float(self.degrees_of_survey)/degrees_of_whole_sky
171
172
```

```
import numpy,copy

class SO():
    def __init__(self):#This class can only be inherited from
        pass

    def drawPSFandSB(self,band):
        dat=self.stochasticobservingdata[band]
        k=numpy.random.randint(len(dat[:,0]))
        return dat[k,0],dat[k,1]

    def CalculateETSB(self,sbs,band):
        et=self.exposuretimes[band]*(len(sbs)*(1./self.nexposures))
        sbf=10**(-(sbs)/2.5)
        sbf=sbf.mean()
        sb=-2.5*numpy.log10(sbf)
        return et,sb

    def PSFfloor(self,a=[],dummy=None):
        #function that encodes the seeing strategy
        mode=self.strategy
        x=self.strategyx
        if mode == "absolute":
            if x==0:
                return 10
            else: return x
        if mode == "percentile":
            import scipy.stats
            return stats.scoreatpercentile(a,x)
        if mode == "best":
            a=numpy.sort(a)
            return a[int(x)]
        if mode == "resolve":
            #return (self.fos*self.bl[1])*self.pixelsize)
            try:
                if
self.bfac*(self.bl[1])**2-self.rfac*(self.rs[1])**2<0:
                    floor1=0
                else:

floor1=(self.bfac*(self.bl[1])**2-self.rfac*(self.rs[1])**2)**0.5
            except FloatingPointError:
                floor1=0 # this should never get called

            try:
                floor2=self.rs[1]*self.magnification[1]
            except KeyError:
                floor2=999

```

```
48
49              floor=numpy.min([floor1,floor2])
50
51              return (floor)*self.pixelsize
52
53          if mode == "resolveclever":
54              print "warning: the resolveclever code isn't finsihed"
55              (self.fos*self.bl[1]*self.pixelsize)
56              numpy.sort(a)
57              la=len(a)
58              b=a[a<(self.fos*self.bl[1]*self.pixelsize)]
59              lb=len(b)
60
61              #definitely include seeings less than the source size:
62              defoin=b[b>self.rs[1]]
63
64              return a[int(x)-1]
65
66              return (self.fos*self.bl[1]*self.pixelsize)
67
68
69      def
…   stochasticObserving(self,mode="MP",seeingstrategy="absolute",stratfloor=
…   10,SOdraw=[],musthaveallbands=False):
70          psfs={}
71          sbs={}
72          psfs2={}
73          sbs2={}
74          worstacceptedpsfband={}
75          worstacceptedpsf=0
76          for band in self.bands:
77              worstacceptedpsfband[band]=0
78              if SOdraw==[]:
79                  psfs[band]=numpy.zeros(self.nexposures)
80                  sbs[band]=numpy.zeros(self.nexposures)
81                  psfs2[band]=numpy.zeros(self.nexposures)
82                  sbs2[band]=numpy.zeros(self.nexposures)
83                  for i in range(self.nexposures):
84                      a,b=self.drawPSFandSB(band)
85                      psfs[band][i]=a
86                      sbs[band][i]=b
87                      psfs2[band][i]=a*1
88                      sbs2[band][i]=b*1
89                  self.SOdraw=[psfs2,sbs2]
90                  psffloor=self.PSFfloor(psfs[band],dummy=band)
91              else:
92                  psfs[band]=SOdraw[0][band]
93                  sbs[band]=SOdraw[1][band]
94                  psffloor=self.PSFfloor(psfs[band],dummy=band)
```

```
95
96              for i in range(len(psfs[band])):
97                  if psfs[band][i]<psffloor and
…  psfs[band][i]>worstacceptedpsf:
98                      worstacceptedpsf=psfs[band][i]
99                  if psfs[band][i]<psffloor and
…  psfs[band][i]>worstacceptedpsfband[band]:
100                     worstacceptedpsfband[band]=psfs[band][i]
101
102             sbs[band]=sbs[band][psfs[band]<psffloor]
103             psfs[band]=psfs[band][psfs[band]<psffloor]
104             if len(psfs[band][psfs[band]<psffloor])==0:
105                 sbs[band]=numpy.array([0])
106                 psfs[band]=numpy.array([0.01])
107
108         #since images need to have same psf, we use the worst accepted:
109         #for band in self.bands:###
110         #    print band,worstacceptedpsfband[band],",",###
111         #print worstacceptedpsfband###
112         for band in self.bands:
113             if mode=="1P":
114                 self.seeing[band]=worstacceptedpsf
115             if mode=="MP":
116                 self.seeing[band]=worstacceptedpsfband[band]
117             if self.seeing[band]==0:continue
118
119             self.psfscale[band]=self.seeing[band]/2.355
120             self.psf[band]=
…  numpy.exp(-0.5*self.r2/(self.psfscale[band]/self.pixelsize)**2)
121             self.psf[band]/=numpy.sum(self.psf[band])
122             self.psfFFT[band]=None
123
124         #print self.seeing###
125
126         #now calculate new exposuretimes and skybrightnesses
127         for band in self.bands:
128
…  self.ET[band],self.SB[band]=self.CalculateETSB(sbs[band],band)
129
130
131         self.seeingtest="Fail"
132         for band in self.bands:
133             if self.SeeingTest(1,band) ==True:
134                 self.seeingtest="Pass"
135         if musthaveallbands:self.seeingtest="Pass"
136
137
138     def SeeingTest(self,src,band):
139         if
```

```
139… self.bfac*(self.bl[src])**2<(self.rfac*(self.rs[src]))**2+(self.seeing[
   … band]/self.pixelsize)**2:
140                 return False
141             else:
142                 return True
143
```

```python
import numpy

class S2N():
    def __init__(self):#This class can only be inherited from
        pass

    def imageRegions(image,sig,sigfloor=0.5):
        image[image/sig<significancefloor]=0
        masks, multiplicity = ndimage.measurements.label(image)
        labels=numpy.arange(1, multiplicity+1)


    def SNfunc(self,data,sig,significancefloor=0.5):
        D=data.ravel()
        S=sig.ravel()

        args=numpy.argsort(-D/S)
        D=numpy.take(D,args)
        S=numpy.take(S,args)
        Dsum=numpy.cumsum(D)
        Ssum=numpy.cumsum(S**2)**0.5
        SN=(Dsum/Ssum).max()

        #regional SN
        import scipy.ndimage as  ndimage
        data[data/sig<significancefloor]=0
        masks, multiplicity = ndimage.measurements.label(data)
        labels=numpy.arange(1, multiplicity+1)
        SNs=numpy.zeros(multiplicity+1)
        SNs[0]=SN
        for i in range(multiplicity):
            D=data[masks==i+1].ravel()
            S=sig[masks==i+1].ravel()
            args=numpy.argsort(-D/S)
            D=numpy.take(D,args)
            S=numpy.take(S,args)
            Dsum=numpy.cumsum(D)
            Ssum=numpy.cumsum(S**2)**0.5
            SNi=(Dsum/Ssum).max()
            SNs[i+1]=SNi
        SNs=-numpy.sort(-SNs)
        return SNs

    def SourceMetaData(self,SNcutA=15,magcut=3,SNcutB=[10,8]):
        self.mag={}
        self.msrc={}
        self.bestband={}
        self.passfail={}
        self.resolved={}
```

```
50
51          for src in self.sourcenumbers:
52              self.resolved[src]={}
53              SNr={}
54              self.mag[src]=self.magnification[src]
55              self.msrc[src]={}
56              for band in self.bands:
57                  if self.seeing[band]!=0:
58
…   self.msrc[src][band]=self.totallensedsrcmag[src][band]
59                      if
…   self.mag[src]*self.rs[src]>(self.seeing[band]/self.pixelsize):
60                          self.resolved[src][band]=True
61                          SNindex=0
62                      else:
63                          self.resolved[src][band]=False
64                          SNindex=2
65                      try:
66                          SNr[band]=self.SN[src][band][SNindex]
67                      except IndexError: SNr[band]=0
68                      except KeyError: SNr[band]=0
69
70                  else:
71                      self.SN[src][band]=[0,0,0]
72                      self.msrc[src][band]=[99]
73                      self.resolved[src][band]=False
74                      SNr[band]=0
75
76
77              self.bestband[src],dummy = max(SNr.iteritems(), key=lambda
…   x:x[1])
78
79              self.passfail[src]=False
80              try:
81                  if self.SN[src][self.bestband[src]][2]>min(SNcutB) and \
82                     self.SN[src][self.bestband[src]][1]>max(SNcutB):
83                      self.passfail[src]=True
84                      ltype=1
85              except IndexError:
86                  pass
87
88
89              try:
90                  #print
…   self.SN[src][self.bestband[src]][0],SNcutA,self.mag[src],magcut,self.
…   resolved[src][self.bestband[src]]
91                  if self.SN[src][self.bestband[src]][0]>SNcutA and \
92                     (self.mag[src]>magcut) and
…   self.resolved[src][self.bestband[src]]:
```

```
93                      self.passfail[src]=True
94                      ltype=2
95              except IndexError:
96                  pass
97
98              if  self.SeeingTest(src,self.bestband[src]) ==False:
99                  self.passfail[src]=False
100
101         #debugger
102         #for src in self.sourcenumbers:
103         #    print
…   src,self.passfail[src],self.SeeingTest(src,self.bestband[src]),self.SN[
…   src][self.bestband[src]][0],self.resolved[src][self.bestband[src]],self.
…   mag[src],ltype
104
105         return self.mag,self.msrc,self.SN,self.bestband,self.passfail
106
107 #=========================================================================
…   ====
108     def
…   RingFinderSN(self,bands=["g_SDSS","i_SDSS"],repair=True,mode="
…   crossconvolve",SNcutA=15,magcut=3,SNcutB=[10,8],runringfinder=False,
…   mustbeseen=False):
109         self.rfpf={}
110         for src in self.sourcenumbers:
111             self.SNRF[src]=[0]
112             self.rfpf[src]=False
113         try:
114             if self.seeing[bands[0]]==0:
115                 return self.rfpf,self.SNRF
116         except KeyError:
117             return self.rfpf,self.SNRF
118         try:
119             if self.seeing[bands[1]]==0:
120                 return self.rfpf,self.SNRF
121         except KeyError:
122             return self.rfpf,self.SNRF
123         if mode=="crossconvolve":
124
…   seeing=(self.seeing[bands[1]]**2+self.seeing[bands[0]]**2)**.5
125             for band in bands:
126                 self.psfFFT[band]=None
127                 self.psfscale[band]=seeing/2.355
128                 self.psf[band]=
…   numpy.exp(-0.5*self.r2/(self.psfscale[band]/self.pixelsize)**2)
129                 self.psf[band]/=numpy.sum(self.psf[band])
130             self.ObserveLens(bands=bands)
131         else: seeing=self.seeing[bands[0]]
132
```

```
133         self.seeing["RF"]=seeing
134 
135         seen=False
136         for src in self.sourcenumbers:
137             if self.SeeingTest(src,"RF"):
138                 seen=True
139         if mustbeseen:
140             seen=True
141 
142         if seen==False:
143             return [self.rfpf,self.SNRF]
144 
145         assert (self.psf[bands[0]]-self.psf[bands[1]]).sum()==0, "psf
    missmatch - can't run ringfinder"
146 
147         B=self.image[bands[0]]
148         R=self.image[bands[1]]
149         sB=self.sigma[bands[0]]
150         sR=self.sigma[bands[1]]
151 
152         r=self.r2**0.5
153         r*=self.pixelsize
154         mask=((r<2.7) & (r>0.5))
155 
156         alpha=B[mask].sum()*1./R[mask].sum()
157 
158         self.D=B-alpha*R
159         self.S=(sB**2+(alpha*sR)**2)**.5
160         self.fakeResidual[0]["RF"]=self.D
161         for src in self.sourcenumbers:
162 
    self.SNRF[src]=self.SNfunc(self.convolvedsrc[src]["g_SDSS"]-alpha*self.
    convolvedsrc[src]["i_SDSS"],self.S)
163 
    d=self.convolvedsrc[src]["g_SDSS"]-alpha*self.convolvedsrc[src]["i_SDSS"
    ]
164             d+=(numpy.random.randn(self.side,self.side)*(self.S))
165             self.fakeResidual[src]["RF"]=d
166             if self.mag[src]*self.rs[src]>(seeing/self.pixelsize):
167                 self.resolved[src]["RF"]=True
168             else:
169                 self.resolved[src]["RF"]=False
170 
171             self.rfpf[src]=False
172             try:
173                 if self.SNRF[src][2]>min(SNcutB) and \
174                    self.SNRF[src][1]>max(SNcutB):
175                     self.rfpf[src]=True
176             except IndexError: pass
```

```
177             try:
178                 if self.SNRF[src][0]>SNcutA \
179                    and self.mag[src]>magcut \
180                    and
…  self.mag[src]*self.rs[src]>(seeing/self.pixelsize):
181                     self.rfpf[src]=True
182             except IndexError: pass
183             if self.SeeingTest(src,"RF")==False:
184                 self.rfpf[src]=False
185                 self.passfail[src]=False
186 
187             if runringfinder:
188                 import RingFinder
189                 RF=RingFinder.RingFinder(B,R,sB,sR,self.pixelsize,
190                                          self.zeromagcounts["g_SDSS"],
191                                          self.zeromagcounts["i_SDSS"])
192                 RFo=RF.ringfind()
193                 self.D=RF.D*1
194                 return RFo,self.rfpf,self.SNRF
195         return self.rfpf,self.SNRF
196 
```

```
from __init__ import *
import cPickle
#import pyfits     .. this was not commented out in original
import sys,os
import pylab as plt
import glob

params = {
   'axes.labelsize': 14,
   'text.fontsize': 14,
   'legend.fontsize': 10,
   'xtick.labelsize': 10,
   'ytick.labelsize': 10,
   'text.usetex': False,
    'figure.figsize': [6, 4]
   }
plt.rcParams.update(params)

sourcepops=["lsst"]

experiment="Euclid"
#experiment="CFHT"
#experiment="LSST"
#experiment="DES"

if len(sys.argv)>1:
    experiment=sys.argv[1]

surveystoread=[]
if experiment=="Euclid":
    surveystoread+=["Euclid"]
elif experiment=="CFHT":
    surveystoread+=["CFHT"]
elif experiment=="CFHTa":
    surveystoread+=["CFHTa"]
elif experiment=="DES":
    surveystoread+=["DESc"]
    surveystoread+=["DESb"]
    surveystoread+=["DESa"]
elif experiment=="LSST":
    surveystoread+=["LSSTc"]
    surveystoread+=["LSSTb"]
    surveystoread+=["LSSTa"]
else:
    surveystoread=[str(experiment)]
    experiment=experiment[:-1]


for survey in surveystoread:
```

```
50    for sourcepop in sourcepops:
51      if survey[-2]=="a":
52          surveyname=survey[:-1]+"_full_coadd"
53      elif survey[-2]=="b":
54          surveyname=survey[:-1]+"_best_epoch"
55      elif survey[-2]=="c":
56          surveyname=survey[:-1]+"_optimal_coadd"
57      else:
58          surveyname=survey
59      filename="%s_%s_lists.pkl"%(survey,sourcepop)
60      lensparsfile="lenses_%s.txt"%survey
61      f=open(lensparsfile,"w")
62      print
63      #os.system("rm %s"%filename) #this line resets the read-in
64      bl={}
65      zs={}
66      zl={}
67      sigl={}
68      ql={}
69      rs={}
70      ms={}
71      mag={}
72      weights={}
73      for key in ["resolved","rfpf"]:
74          bl[key]=[]
75          zs[key]=[]
76          rs[key]=[]
77          ms[key]=[]
78          zl[key]=[]
79          sigl[key]=[]
80          ql[key]=[]
81          mag[key]=[]
82          rs[key]=[]
83          weights[key]=[]
84
85      if experiment=="CFHT":
86        frac=42000.*1./150.
87        bands=["g_SDSS","r_SDSS","i_SDSS"]
88
89      if experiment=="CFHTa":
90        frac=42000.*1./150.
91        bands=["g_SDSS","r_SDSS","i_SDSS"]
92
93      elif experiment=="Euclid":
94        frac=42000.*1./15000.
95        bands=["VIS"]
96
97      elif experiment=="DES":
98        frac=42000.*1./5000.
```

```
99        bands=["g_SDSS","r_SDSS","i_SDSS"]
100
101     elif experiment=="LSST":
102       frac=42000.*1./20000.
103       bands=["g_SDSS","r_SDSS","i_SDSS"]
104
105
106
…   filelist=glob.glob("LensStats/%s_%s_Lens_stats_*.pkl"%(experiment,
…   sourcepop))
107
108     chunki=0
109     ilist=[]
110     print survey
111     for chunk in filelist:
112         print chunki
113         chunki+=1
114         f2=open(chunk,"rb")
115         fracsky,sspl=cPickle.load(f2)
116         fract=frac*fracsky
117         f2.close()
118         I=0
119         for i in sspl.keys():
120             if i in ilist:
121                 continue
122             else:
123                 try:
124                     sspl[i]["seeing"][survey]
125                 except KeyError:
126                     continue
127                 f.write("%.2f "%sspl[i]["zl"])
128                 f.write("%.2f "%sspl[i]["zs"][1])
129                 f.write("%.2f "%sspl[i]["b"][1])
130                 f.write("%.2f "%sspl[i]["sigl"])
131                 f.write("%.2f "%sspl[i]["ql"])
132                 f.write("%.2f "%sspl[i]["rl"]["g_SDSS"])
133                 for band in bands:
134                     f.write("%.2f "%sspl[i]["ml"][band])
135                 f.write("%.2f "%sspl[i]["rl"]["g_SDSS"])
136                 f.write("%.2f "%sspl[i]["xs"][1])
137                 f.write("%.2f "%sspl[i]["ys"][1])
138                 f.write("%.2f "%sspl[i]["qs"][1])
139                 f.write("%.2f "%sspl[i]["ps"][1])
140                 f.write("%.2f "%sspl[i]["rs"][1])
141                 f.write("%.2f "%sspl[i]["mag"][1])
142                 for band in bands:
143                     f.write("%.2f "%sspl[i]["seeing"][survey][band])
144                     f.write("%.2f "%sspl[i]["SN"][survey][1][band][0])
145                 if survey!="Euclid":
```

```
146                     f.write("%.2f "%sspl[i]["rfsn"][survey][1][0])
147                 f.write("\n")
148
149
150                 ilist.append(str(i))
151                 if sspl[i]["pf"][survey][1]==False:continue
152
153                 try:
154                     bb=sspl[i]["bestband"][survey][1]
155                     #print sspl[i]["seeing"][survey][bb]
156                     #print sspl[i]["mag"][1]*sspl[i]["rs"][1],
157                     try:
158                         (sspl[i]["b"][1]**2-sspl[i]["rs"][1]**2)**0.5
159                     except FloatingPointError: print 0
160                 except KeyError:
161                   pass
162                 try:
163                   if
…  sspl[i]["resolved"][survey][1][sspl[i]["bestband"][survey][1]]:
164                     bb=sspl[i]["bestband"][survey][1]
165                     if sspl[i]["mag"][1]<3:continue
166                     if sspl[i]["SN"][survey][1][bb][0]<20:continue
167
168                     bl["resolved"].append(sspl[i]["b"][1])
169                     weights["resolved"].append(1./fract)
170                     zs["resolved"].append(sspl[i]["zs"][1])
171                     rs["resolved"].append(sspl[i]["rs"][1])
172                     zl["resolved"].append(sspl[i]["zl"])
173                     sigl["resolved"].append(sspl[i]["sigl"])
174                     ql["resolved"].append(sspl[i]["ql"])
175                     mag["resolved"].append(sspl[i]["mag"][1])
176                     ms["resolved"].append(sspl[i]["ms"][1]["g_SDSS"])
177
178                     if sspl[i]["rfpf"][survey][1]:
179                         if sspl[i]["rfsn"][survey][1][0]<20:continue
180                         if
…  sspl[i]["resolved"][survey][1]["RF"]==False:continue
181
182                         if experiment=="CFHT" or experiment=="CFHTa":
183                             if sspl[i]["zl"]>1:continue
184                             if sspl[i]["zl"]<0.2:continue
185                             if sspl[i]["ml"]["i_SDSS"]<17:continue
186                             if sspl[i]["ml"]["i_SDSS"]>22:continue
187
188                         bl["rfpf"].append(sspl[i]["b"][1])
189                         weights["rfpf"].append(1./fract)
190                         zs["rfpf"].append(sspl[i]["zs"][1])
191                         rs["rfpf"].append(sspl[i]["rs"][1])
192                         zl["rfpf"].append(sspl[i]["zl"])
```

```
                        sigl["rfpf"].append(sspl[i]["sigl"])
                        ql["rfpf"].append(sspl[i]["ql"])
                        mag["rfpf"].append(sspl[i]["mag"][1])
                        ms["rfpf"].append(sspl[i]["ms"][1]["g_SDSS"])


                except KeyError:
                  pass
    f.close()

    if survey[-2]=="a":
        surveyname=survey[:-1]+" (full coadd)"
    elif survey[-2]=="b":
        surveyname=survey[:-1]+" (best single epoch imaging)"
    elif survey[-2]=="c":
        surveyname=survey[:-1]+" (optimal coadd)"
    else:
        surveyname=survey

    print survey, "will find",
    print numpy.sum(numpy.array(weights["resolved"]).ravel()),
    print "lenses assuming poisson limited galaxy subtraction in all
bands, or",
    print numpy.sum(numpy.array(weights["rfpf"]).ravel()),
    print "lenses in the g-i difference images"

    f=open(filename,"wb")
    cPickle.dump([weights,bl,zs,rs,ms,zl,sigl,ql,mag],f,2)
    f.close()



bson=numpy.array([2.66,1.24,1.27,2.39,1.41,1.27,1.00,1.3,1.0,1.19,1.22,1
.36,1.76,1.19,1.29,1.56,1.04,0.85,1.10,1.23,1.16,0.93,1.03,1.4,0.74,1.21
,1.14,1.74,2.03,1.23,2.55,1.05,1.51,4.36,0.94,0.93,3.11,1.79,0.96,1.40,1
.3,0.81,1.95,1.66,1.55,1.07,1.06,1.38,0.52,2.16,1.40,1.44])
plt.hist(bson,bins=numpy.linspace(0,3,16),weights=bson*0+220./len(bson),
fc="grey",alpha=0.6)
a,b=numpy.histogram(bl["rfpf"],bins=numpy.linspace(0,3,31),weights=
weights["rfpf"])
a*=2#double for finer bins
plt.plot(b[:-1]+(b[1]-b[0])/2.,a,c="k",lw=3,ls="dashed")
plt.xlabel(r"$\Theta_\mathrm{E}$ (arcsec)")
plt.ylabel(r"Lenses per $\Theta_\mathrm{E}$ bin")
plt.tight_layout()
plt.show()

```

A.2 Mapping the Dependencies

The following is an outline of the workflow of the model, undertaken as a first step to understanding and mapping out the dependencies within the code. Throughout the project, this served as a handy reference for following through routines and for subsequently identifying any areas of coding that might require adjusting or amending.

Outline of Workflow

Module abbreviations:

MakeLensPop (MLP)	*PopulationFunctions (PFs)*	*ModelAll (MAll)*	*FastLensSim (FLS)*
*SBModels (SBM)**	*SBProfiles(SBP)*	*Surveys (Sur)*	*StochasticObserving (Sto)*
SignatoNoise (SN)	*MakeResults (MRs)*	**note capital **M**: cloning gives only SB**m**odels*	

****

MakeLensPop

246 (MLP) Creates **D** which is an object of class **Distance**, the initialisation of which creates distance attributes for use within the code

247 (MLP) Creates **Lpop** which is an object of class **LensPopulation**
*main parameters are zlmax = 2, sigfloor = 100, reset = **True***

8 (MLP) **LensPopulation** inherits class properties from **LensPopulation_**
144 (PFs) *which inherits class properties from **Population***
123 (PFs) *which inherits class from **RedshiftDependentRelation***

16 (MLP) Initialisation of **LensPopulation** runs **beginRedshiftDependentRelation**
12 (PFs) *main parameters include cosmo = [0.3, 0.7, 0.7], reset = **True***

13-29 (PFs) Creates redshift (z) splines Da_spline, Dmod_spline, volume_spline,
and Da_bispline, as reset = **True**, by running **redshiftfunctions** to
31-58 (PFs) create and dump them to a new pickle file
23-27 (PFs) *if reset = **False**, then **beginRedshiftDependentRelation** loads existing redshift splines from pickle file (or runs **redshiftfunctions** to create new one if exception)*
37 (PFs) *creates Da2bins by running **distances.Da** function on z2bins[i] and*
43-46 (PFs) *z2bins[j] but only where j>i otherwise default **zeros***
(nb. z2bins[i], z2bins[j] are in same ascending order)
54 (PFs) *Da2bins is used to create Da_bispline*
116 (PFs) *calculation of dls in Einstein radius calculation calls evaluation (**ev**) of*
78/54 (PFs) *Da_bispline, reading in zl = z1 = z2bins[i] and zs = z2 = z2bins[j]*
118 (PFs) *(if zl > zs this inter/extrapolates to a negative result which => rein = 0)*

17 (MLP) Initialisation of **LensPopulation** then runs **beginLensPopulation**
157 (PFs) Sets parameter reset to **True**
159-174 (PFs) This routine is not executed as reset is **True**
176 (PFs) Runs **lenspopfunctions** to create lens population splines and dump them into a new pickle file (as reset = **True**)
*if reset not **True**, then loads splines in from existing pickle file*
179 (PFs) **lenspopfunctions** runs **Psigzspline**

183-219 (PFs) which creates splines for density functions
this section corresponds to expression (3) in article

186 (PFs) Sets zlbins = 200 units from 0 to zlmax ; dzl = bin size (zlmax/200)
187 (PFs) Sets sigbins = 400 units from sigfloor to 400
189 (PFs) Initialises dNdz (used in spline) with zeroes in an array same size as zlbins
lines 183-223 follow expression (3) in article
modified Schecter function => dn = $\Phi(\sigma)d\sigma$ => dn/dσ = Φ

193-202 (PFs) Commences 'for' loop to create and build up dNdz array (200 elements)
195 (PFs) Defines dphidsiggivenz, based on function **phi**
*should really read 'd**n**dsiggivenz'*
19-35 (MLP) Inherits this definition of function **phi** (=> redshift independent, as per article)
293-305 (PFs) *cf. other definition of function **phi** (=> redshift dependent) not used*

196 (PFs) Defines phisigspline based on dphidsiggivenz,
*so this should really read '**n**sigspline'*
197 (PFs) *tot = dn (= dN)*

195-200 (PFs) *independence of **phi** from redshift => same results returned for all values of **i** in these lines of the 'for' loop (also results in 195 = 217, & 199 = 219)*

202 (PFs) Derives dNdz as dN/dv.dv/dz
(dN given is per unit volume; dv/dz is the cosmology 'comoving volume')

209 (PFs) *this is the cumulative distribution function giving the relationship between a specified z^* and N with z <= z^*.*
204 (PFs) (*read Nofzcdf as N(z); not as 'number of z' cdf)*

211 (PFs) Creates dNdzspline as spline between zlbins and dNdz

212 (PFs) Obtains value for N by integrating dn/dz up to zmax
(cf. Ndeflectors calculation in line 286)

180 (PFs) **lenspopfunctions** runs **Colourspline**
241 (PFs) *to determine colour for z and band*
[allows K-correction/galaxy evolution]

181 (PFs) **lenspopfunctions** runs **lensPopSplineDump**
246 (PFs) *to pickle all the splines created by **Psigzspline** and **Colourspline** functions*

248 (MLP) Runs function **Ndeflectors** on **Lpop** to create Ndeflectors, which is the
286 (PFs) number of deflectors in redshift up to z=2
Ndeflectors is an integer (debug shows 1,102,981,692)
286 (PFs) *main parameters are z = 2, fsky = fraction of sky (default=1)*
289 (PFs) Derives Ndeflectors by integrating dNdzspline between z=0 and z=2
291 (PFs) *N is independent of the chosen source population (eg. 'lsst')*

202 (PFs) *Arguably the model should be relying on a fixed number for Ndeflectors regardless of cosmology. Since any change to cosmology will feed through into Ndeflectors through the comoving volume (dVol/dz), then presumably either (i) the comoving volume should be hardcoded for the purpose of calculating **Ndeflector**s, or (ii) **Ndeflectors** itself should be hardcoded*

249 (MLP) Creates **L** which is an object of class **LensSample**
55 (MLP) *main parameters are sourcepop='lsst', sigfloor=100, zlmax = 2, reset = **False**, and flat LCDM cosmology*
63 (MLP) *sets sourcepopulation = sourcepop = 'lsst'*

64 (MLP) **LensSample** initialisation creates sub-class object **D=Distance**, the initialisation of which creates distance attributes for use within the code
60 (MLP) *by default D = None, so this is a duplication of the routine called in line 244 (MLP)*

68 (MLP) **LensSample** initialisation creates sub-class object **L=LensPopulation**
8 (MLP) *main parameters are zlmax = 2, reset = **False***
*(caution: **L** now used for LensPopulation not LensSample)*
8-17 (MLP) Initialises sub-class object **L=LensPopulation** as for **Lpop** above, and includes **beginRedshiftDependentRelation** and **beginLensPopulation** functions;
*difference between line 247 (MLP) and line 249->68 (MLP) is that reset = **True** and **False** respectively - but it is set to **True** in line 157 (PFs) so routine in 158-176 (PFs) is the same; hence, **redshiftsplines** pkl file is unaffected but a new **lenspopsplines** pkl file is created*

70 (MLP) **LensSample** initialisation creates sub-class object **S=SourcePopulation**
61/70 (MLP) *main parameters include population = sourcepop = 'lsst', reset = **False***
38-43 (MLP) **SourcePopulation** initialisation runs **beginRedshiftDependentRelation**
appears to be no difference in parameters between lines 249->68->16 (MLP) and lines 70->43 (MLP), so not clear why this is run

47 (MLP) **SourcePopulation** initialisation runs function **loadlsst**
396-419 (PFs) **loadlsst** function loads in LSST data (= source galaxy parameters) from pickle file and creates corresponding variables (stellar and halo masses, magnitudes and redshifts)

72 (MLP) **LensSample** initialisation creates sub-class object **E=EinsteinRadiusTools**
103 (PFs) **EinsteinRadiusTools** initialisation runs **beginRedshiftDependentRelation**
104/12 (PFs) *appears to be no difference in parameters between lines 249->68->16 (MLP) and lines 72 (MLP)->104 (PFs), so not clear why this is run*

250 (MLP) **Generate_Lens_Pop** runs on **LensSample** (**L**) to draw foreground and
78-103 (MLP) background galaxy populations (ie. potential lenses and sources)

*main parameters include Ndeflectors (an integer), nsources = 1, prunenonlenses = **True**, firstod (first over-density = difference from average density) = 1*

****

Generate_Lens_Pop

79-101 (MLP) Displays progress on screen
N is total number of deflectors
M is number of deflectors yet to be processed in routine
n is batch of deflectors for current pickle file
81 (MLP) *note 'prune **non**-lenses' parameter*
85/100 (MLP) *multiplying an array by *1 breaks the link between the two variables*

86/87 (MLP) Initialises a deflector counter l and an idealised lens counter l2, both with value = -1

102 (MLP) **Generate_Lens_Pop** runs **drawLensPopulation** on **LensPopulation** (**L**) to
319 (PFs) return arrays of zl (lens redshift), sigl (lens sigma), ml (lens magnitude), rl (lens radius), ql (lens ellipticity)
main parameter is number = n

320/261 (PFs) Runs function **draw_zsig** which in turn runs functions **draw_z** and **draw_sigma**

262 (PFs) Derives zl from **draw_z** function
main parameter is N = number
248 (PFs) **draw_z** function produces an array (of length N = no of deflectors) random numbers (from 0 to 1) and then for each draws a value of zl from
209 (PFs) cdfdNdzasspline using those random numbers as Nofzcdf
cdfNdzasspline returns the zl for which the likelihood is that this number N of deflectors will have redshift up to & including that zl

251 (PFs) Derives sigl from **draw_sigma** function
263 (PFs) *main parameter is z = zl (as returned from **draw_z** routine above)*
254 (PFs) *nozdependence = **True** from line 33 (MLP)*
255 (PFs) **draw_sigma** function produces an array (of length = no of deflectors) random numbers (from 0 to 1) and then for each draws a value of sigl using
255 (PFs) cdfNdsigz0asspline which returns the sigl for which the likelihood is that this (random) number of deflectors will have sigl up to & including that sigl

321 (PFs) Derives ql from **draw_flattening** (sigl=sigl) function
308-317 (PFs) *this section corresponds to expression (4) in article*
a Rayleigh distribution is used, and note that q is an array
310 (PFs) *note typo' in expression (4) in article (coefficient x)****
314 (PFs) Truncates values of q array elements so q[] >0.2 and <=1
(eg. *q[q<0.2] => array of q with elements <0.2)*

322 (PFs) Mr (absolute R_band magnitude) and r_phys_nocol (**observed** R_band size)
266 (PFs) are derived by running **EarlyTypeRelations** function on **LensPopulation (L)**
266 (PFs) *caution: narrative says z dependence not encoded*
268-9 (PFs) *Hyde and Bernardi used to obtain Mr based on sigma=sigI*
273-4 (PFs) *R = **rest frame** R_band size, which is determined from Mr (or sigI)*
279 (PFs) *r_phys_nocol = 10^R*

323 (PFs) Creates mI (apparent magnitude), rI (apparent size), and r_phys as dictionaries with bands as the keys

327 (PFs) Sets r_phys equal to r_phys_nocol for each band
(=> each band has the same value)
narrative allows for addition of a colour function here

328-330 (PFs) For each band that is not 'VIS', the value of mI is determined from the
127 (PFs) function **draw_apparent_magnitude**
main parameters are M=Mr, z=zI, band=band
283, 234 (PFs) ***draw_apparent_magnitude*** calls function ***colour*** *which uses coloursplineI in each band for each z*

331 (PFs) rI has a value for each band which is determined from the function
137 (PFs) **draw_apparent_size**
(= r_phys divided by angular diameter distance Da converted into arcsecs)
main parameters are r_phys = r_phys[band], z=zI
('apparent size' is an angle)
139 (PFs) *nb. 1 radian = 206,264 arcsecs*

103 (MLP) **Generate_Lens_Pop** runs **drawSourcePopulation** on **SourcePopulation (S)** to return arrrays for zs, ms, xs, ys, qs, ps, rs, mstar, mhalo
445 (PFs) *main parameters are number=n*nsources,*
*sourceplaneoverdensity=firstod, returnmasses=**True**,*
sourceplaneoverdensity is density excess in source plane over average;
nsources = 1 => no of sources = no of deflectors

446 (PFs) Creates **source_index** as an array of 3 x (n*nsources) random elements (each of value between zero and the length of elements in zc)
debug shows len(zc) = 34,331 (=> some indices repeated)
448 (PFs) The number of elements in **source_index** is then restricted (sliced) to the first n elements
recall n is number (of deflectors) in the pickle file
447 (PFs) *the restriction on index values (>10, <0.05) is commented out*

449 (PFs) Creates zs, Mvs as arrays populated from zc and Mv according to key **source_index** (=> zs and Mvs each contain an array of n values drawn from n random positions of the larger arrays zc and Mv)
eg. zs = zc[array(3,7,2,6,5...)] => zs[0] = zc[3], zs[1] = zc[7] ...
406 (PFs) *zc and Mv data are supplied from **loadlsst** function*

451 (PFs) Creates ms as a dictionary with band as keyword, and for each band there
452-456 (PFs) is an array populated from m according to key **source_index** (=> for each band, ms contains an array of n values drawn from n random positions of the larger array of m)

456 (PFs) *if the band is 'VIS', an average value of r_SDSS, i_SDSS and z_SDSS is used*

406 (PFs) *m data is supplied from **loadlsst** function*

458 (PFs) Creates r_phys from the **RofMz** function

main parameters are Mvs and zs

this represents effective radius based on magnitude M and redshift z, with ellipticity drawn from Rayleigh distribution; it is an array (size is same as Mvs and zs arrays).

421 (PFs) *this corresponds to expression (5) in article*

423-426 (PFs) *possible errors in both article and/or code; see separate note*

459 (PFs) Creates rs as an array from the **draw_apparent_size** function

137 (PFs) *main parameters are r_phys = r_phys, z = zs*

460 (PFs) Creates qs as an array from the **draw_flattening** function

main parameter is N=number (=n)

436-443 (PFs) *this section corresponds to the text just prior to expression (5) in article; the routine uses a Rayleigh function (with scale factor = 3) to create 1.5N elements, and then extracts only N elements accepting only values > 0.2 (1.5N elements created to allow for trimming prior to extraction)*

462 (PFs) Creates ps as an array of a number (=n) of random elements in the interval 0 -> 180

corresponds to an array of n random angles up to 180 deg.

476-477 (PFs) Creates xs and ys each as an array of a number (=n) of random elements in the interval -0.5 -> +0.5 multiplied by a

469-471 (PFs) *a is the scaling factor needed to adjust for source density; xs, ys coordinates of (the centre of) a source are drawn from a grid of 4.08 x 4.08 arcsecs as LSST density of 0.06 per sq arcsec => 1 per 16.67 sq arcsecs (ignoring sourceplaneoverdensity) (not to be confused with 50 x 50 'postage stamp' for **pixeval** in 162 FLS)*

480-481 (PFs) Creates mstar_src and mhalo_src as arrays populated from mstar and mhalo according to key **source_index** (=> mstar_src and mhalo_src each contain an array of n values drawn from n random positions of the larger arrays mstar and mhalo)

416-417 (PFs) *data is supplied from **loadlsst** function*

[commands on lines 102 and 103 (MLP) completed]

105 (MLP) *zl1 and zl connection 'broken' by '*1'*

106 (MLP) *sigl and sigl1 connection 'broken' by '*1'*

107-109 (MLP) This routine does not run if nsources = 1; but if there are multiple sources (nsources>1) then the zl & sigl arrays are duplicated for each source (eg. 3 sources => original plus 2 duplicates of zl & sigl arrays)

111 (MLP) Runs the **sie_rein** function to determine the Einstein radius b for each
113-119 (PFs) background and foreground object pair (*i*), based on the corresponding sigl, zl, and zs values
see expression (2) in article & also Obs. Cos. expression (7.26)

112 (MLP) Commencement of a 'for i in range (n)' routine for each background-foreground pair i
88-101 (MLP) *n = **100**,000 (or residual) puts the total no of deflectors into chunks of **100**,000 (or residual) for the 'for i in range' routine; enables 'time left' progress to be shown*
114 (MLP) Creates an object lens that is an array, with each element l as a dictionary
86/113 (MLP) *l is augmented by 1 on each pass*

115-118 (MLP) Tests each background-foreground pair i to see if xs, ys is within Einstein radius, and if so, sets Lens? in lens[l] to **True** (otherwise False)
this section corresponds to expression (6) in article

120-158 (MLP) Runs a routine to populate remaining elements of lens[l] with data for each background-foreground pair i

122-123 (MLP) Populates the zl and sigl elements of lens[l] with values extracted from zl, sigl arrays indexed by i key
124/146 (MLP) *nsources = 1 => 'j in range (nsources)' routines only execute once; should be able to accommodate multiple sources (populating extended array elements of specific lens[l] with additional source properties) BUT it seems 'Lens' identifier depends only on 'first' background galaxy (ie. second & third background objects may be reported as sources even if they are not).*

125/150 (MLP) *note duplication*
126/151 (MLP) *note duplication*

157/158 (MLP) *note discrepancy in 'mhalo' and 'mstar' elements ***

161-2 (MLP) If lens[l]['lens?'] is **True** (=> background-foreground pair represent source and lens system), and prunenonlenses is **True** (default), then
163 (MLP) l2 is augmented by 1 and
165 (MLP) data from lens[l] is copied into a new object called reallens[l2] and
167-8 (MLP) lens is deleted and a new lens initialised
170-171 (MLP) l2 is displayed if l2 modulo 1,000 = 0

86/113 (MLP) *think of l2 as true lens counter, while l is a background-foreground counter (regardless of whether a lens) and serves as an index for each pair*

173-181 (MLP) if save is **True** (default) and l2 modulo 10,000 = 0 then dump reallens to a pickle file, delete reallens, and initialise a new reallens
*lens-source pairs are dumped into pickle files in batches of **10**,000*

183-185 (MLP) If lens[l]['lens?'] is **False**, and prunenonlenses is **True** (default), then lens is deleted and a new lens initialised

161-185 (MLP) Return to 'for i in range' routine (line 112) if i still in range, otherwise return to 'while M > 0' routine (line 58) for next chunk of **100**,000 (or residual)

186-191 (MLP) Runs after completion of 'while M > 0' routine; if save is **True** (default) then dumps reallens to a pickle file
*pickle file will contain 'residual' (l2 modulo **10**,000) lens-source pairs*

193-200 (MLP) If prunenonlenses = **False** then runs routine for pickling non-lens system data

****

ModelAll.py

1 (MAll) Initialisation imports modules **PopulationFunctions**, **MakeLensPop**, **Surveys**, and **FastLensSim**
7 (MAll) *sigfloor default set at 200*

9 (MAll) Creates **L** as an object of class **LensSample**
*(initialisation of **LensSample/LensPopulation** classes does not draw lenses,*
146-183 (PFs) *but creates splines; lenses are only drawn by **Generate_Lens_Pop** function)*
249 (MLP) *this is a repeat of the routine run in **MakeLensPop***

11 (MAll) Sets experiment as Euclid

12 (MAll) Sets frac = 0.1
this corresponds to fracsky subsequently used in the MRs module, and is used to scale down the number of foreground-background pairs tested by
76-77 (MAll) *the code (eg. for Euclid this is 1,253,000 instead of 12,350,000).*
*code probably intended originally to take a sample of only 0.1 (frac) of the idealised lenses before applying detection criteria (presumably to save processing time). The output is subsequently scaled up again by a factor 10 in the MRs module. But the code does not in fact take 0.1 of the idealised lenses: it takes a sample size of 0.1 of **12,530,000** - this being a hardcoded number. In the case of the standard cosmology, there are about 11.9m*

idealised lenses, so correctly scaling up by 11,900/1,253 = 9.5 is close enough to scaling up by a factor 10.

14 (MAll) Sets a = 20
this is signal-to-noise threshold - see expression (9) in article
15 (MAll) Sets b = 3
this is magnification threshold - see expression (8) in article
17 (MAll) Sets c = 100
18 (MAll) Sets d = 1000

32 (MAll) Sets nsources = 1

35 (MAll) Creates a list called **surveys** with elements corresponding to survey names
38 (MAll) *experiment (survey name) currently set to 'EUCLID'*

54-55 (MAll) Creates and initialises a dictionary S and a dictionary n
56 (MAll) For each survey name, creates a dictionary item with that survey name as the key and the corresponding value of S[survey] as a **FastLensSim** class object for that survey; this object inherits properties from SO (**StochasticObserving.py**) and S2N (**SignaltoNoise.py**)
*(eg. creates S['Euclid'] as a **FastLensSim** class object)*

15-20 (FLS) Begins initialisation of **FastLensSim** object S[survey] by creating survey which is an object of class **Surveys.Survey**
3-170 (Sur) Initialises survey=**Surveys.Survey** by reading in parameters from **Surveys**

5-6 (Sur) *strategy = "resolve", strategyx = 1*

116-128 (Sur) *for EUCLID: pixelsize = 0.1, side = 200, bands = 'VIS', zeropoints = 25.5, zeroexposuretime = 1, sky brightness = 22.2, exposuretimes = 1610,* gains = 1, *seeing = 0.2, nexposures = 4, degrees_of_survey = 20000, readnoise = 4.5*

170 (Sur) End of initialisation for **Surveys.Survey** sets *f_sky = degrees_of_survey /degrees_of_whole_sky*
169 (Sur) *('degrees_of_whole_sky' = 41253)*

22-50 (FLS) Initialisation of **FastLensSim** object S[survey] continues by setting attributes according to parameters read in to **Survey** class object survey (from initialisation of **Surveys.Survey** object)
53-54 (FLS) Sets xl = 99.5, yl = 99.5
=> xl and yl are the central pixel in a 200 x 200 grid
=> the lens is at the origin

55 (FLS) Runs function **IT.coords** to create an x, y grid of sides each of length 200
56 (FLS) Creates r2 as the distance from any pixel to xl, yl (ie. to the origin)
r2 is an array; r2 <=> r^2

57/58 (FLS) Defines attributes bfac and rfac each to return value 2.0

61-92 (FLS) Initialisation of **FastLensSim** ends by running the function **Reset** to initialise all the foreground, background and related parameters
note: this does not affect pickled data (in idealised lens pickle file)

79 (MAll) Commences 'for i in range' loop for each background-foreground pair, which ends at line 241

82 (MAll) Runs the function **LoadLensPop** on **L** to load in data for pair i from the
202-205 (MLP) corresponding idealised lens pickle file
85-93 (MAll) *displays progress*

244 (MAll) *('accepted lens' counter (Si) also displayed at end of run)*

95 (MAll) Runs **lens** function on **L** to create a dictionary lenspars for each deflector-
204 (MLP) source pair i, which contains the properties of that pair read in from the idealised lens pickle file
requires LoadLensPop to be run (line 82)

96-98 (MAll) If lenspars['lens?'] = **False**, then delete that pair's properties and return to the beginning of the 'for i in range' loop for the next pair

183 (MLP) *note: prunenonlenses default setting (**True**) => foreground object (deflector) of pair in pickle file is a lens*

100 (MAll) For the properties (lenspars) for pair i, sets the 'VIS' component of rl equal to the average of the r_SDSS, i_SDSS and z_SDSS values

102-103 (MAll) For each pair, this sets the VIS magnitude for the lens mi = lenspars["ml"] and for the source mi = lenspars["ms"][1] so that it is the average of their respective r_SDSS, i_SDSS and z_SDSS values
keyword = 1 for ms implies single source only
source VIS magnitude (ms[1]['VIS']) is initially empty: see Table 1(d) in article

110-121 (MAll) Initialises empty elements (that are also dictionaries) for additional properties of the deflector-source pair to be inserted into the 'lenspars' dictionary
110-113 (MAll) *note duplication of lenspars[mag] and lenspars[msrc]*

123 (MAll) Sets lastsurvey = "non"

124 (MAll) Commences for survey in surveys loop, which ends at line 222
79 (MAll) *this is within the 'for i in range(nall)' loop*

126 (MAll) For each survey, this runs the function **setLensPars** on S[survey] to create

102-123 (FLS) elements (ie. sets parameters) including those corresponding to the lens properties ml, rl, ql
*main parameters are m = ml, r = rl, q = ql, n = 4, pixelunits = **False,***
*reset = **True**, xb = xp = jiggle = 0*

105-107 (FLS) Default setting of pixelunits = **False** => this converts half-light radius for lens rl
96-98 (FLS) into number of pixels using **trytoconvert** function

111-117 (FLS) Default setting of jiggle = 0 => deltax1 = deltay1 = deltap = 0
119 (FLS) Sersic index for lens (nl) defaults to 4 (=> de Vaucouleurs profile)
113-115 (FLS) *if jiggle <>0, then nl is randomly set with 4 < nl < 8)*

120 (FLS) Creates gal which is an object of class **SBModels.Sersic**
59-67 (SBM) Initialises **SBModels.Sersic** class = > initialises **SBModels.SBModel** class
15 (SBM) *name = '**gal**', and pars = lens parameters (pars is a dictionary)*

20/61 (SBM) Checks parameter names against those in array **_SBkeys**
17 (SBM) *sets parameter amp =1 if not specified*

24/60 (SBM) Runs initialisation for object of class **_baseProfile = SBProfiles.Sersic**

12 (SBP) Initialises object of class **SBProfiles.Sersic**
23 (SBP) Sets NoFreeParams = **False**
according to narrative, this is a "flag to tell the code not to pixeval every step, if nothing changes"
20 (SBP) Sets convolve = **True**

25 (SBM) Creates empty dictionary vmap
29 (SBM) *'**value**' attribute does not exist => always exception, so*
32 (SBM) *always goes to **_setattr_** function, but note also*
33/54 (SBM) ***setPars** function does not run as vmap is empty*

38-51 (SBM) Sets pa (converts from degrees to radians), theta (converts from radians to degrees), and amp (creates amp from logamp) and sets (a dictionary of) other attributes and values for '**gal**' (an object of **SBModels.Sersic class**)

127-131 (MAll) For each survey, this runs the function **setSourcePars** (**FastLensSim**) on
128-152 (FLS) S[survey] to create elements including those corresponding to the source properties b, ms, xs, ys, qs, ps, rs
128 (FLS) *Sersic index n = 1 by default*
32 (MAll) *sourcenumber = nsources = 1 => j = 0; only single source accommodated (recall Einstein radius b depends on source distance)*

130-133 (FLS) Default setting of pixelunits = **False** => this converts source coordinates xs, ys
96-98 (FLS) into number of pixels using **trytoconvert** function

134-135 (FLS) This adjusts xs, ys coordinates by xl, yl (and deltas) respectively to allow

53-54 (FLS) for centralisation of lens
142 (FLS) Creates src[sourcenumber] as a **SBModels.Sersic** class object
main parameters are name = '' src 'sourcenumber' '';
pars = dictionary with x: xs[sourcenumber], y: ys[sourcenumber], q: qs[sourcenumber], pa: ps[sourcenumber], re: rs[sourcenumber], n: ns[sourcenumber]; convolve = 0
*see also routine for **gal** in line 120 (FLS)*

146-152 (FLS) Initialises various parameters

133 (MAll) If this *is* a repeat of the previous survey for that pair i, then jumps to:
146 (MAll) **LoadModel** function (and subsequent routine)

134 (MAll) If this is *not* a repeat of the previous survey for that pair i, then runs a
278 (FLS) function **makeLens** on S[survey] to create an object called model
main parameter is stochasticmode = 'MP'
290 (FLS) *model comprises values returned for galmodel, sourcemodel, model, magnification, and totallensedsrcmag*

278 (FLS) **makeLens** runs a function **stochasticObserving** (from StochasticObserving.py)
279 (FLS) if stochastic = **True** (default)
*main parameters are seeingstartegy = "absolute", mode = "MP", musthaveallbands = **False***

69-129 (Sto) **stochasticObserving** function returns stochastic values for observing conditions, such as point spread function (psf) and sky brightness (sbs)
84 (Sto) ***drawPSFandSB** function draws a pairing of psf and sbs values by taking a*
7-10 (Sto) *random row, **first** column (PSF) value with the same row, **second** column (SB) value from **stochasticobservingdata** (2-D) array of PSF and SB values (for Euclid, this is always 0.17, 22.2 respectively); these values are then used*
89-90 (Sto) *to populate SODraw=[psfs, sbs] and psffloor=**PSFfloor**(psfs,band)*

19-66 (Sto) ***PSFfloor** is a function that "encodes the seeing strategy"; it returns (floor =) psffloor (scaled by pixelsize), which is the maximum value allowed for the 'seeing' by expression (7) in Collett article (or zero if $\theta^2 < r^2$)*
43-49 (Sto) *magnification is {} => **KeyError**, so floor2=999 => floor = floor1*
36-41/44(Sto) *floor1 => non-magnified source; floor2 => magnified source (used for LHS of expression (8) in Collett article)*

96-100 (Sto) **stochasticObserving** function derives worstacceptedpsf/band and
112-117 (Sto) sets S[Euclid].seeing = worstacceptedpsf/band, which is equal to psfs (a
75/77 (Sto) stochastic value of psf) or to zero depending on value of psffloor
if psfs ('seeing') < psffloor (=> acceptable) AND is greater (=> 'worse') than the existing worstacceptedpsf, then replaces the existing worstacceptedpsf with this psf; this corresponds to section 3 in Collet article

129 (Sur) *initialised value of S[Euclid].seeing = 0.2 (cf. Table 1 in article; seeing = 0.18)*

122/129 (Sur) *S[Euclid].SB (skybrightness) = 22.2*
123 (Sur) *S[Euclid].ET (exposuretimes) = 1610*
117 (Sur) *S[Euclid].pixelsize = 0.1*

127-128 (Sto) derives exposuretimes and skybrightness using function **CalculateETSB**

131-142 (Sto) **stochasticObserving** function runs function **SeeingTest** to return seeingtest as **Pass** or **Fail** (worstacceptedpsf/band = > using 'worst' psf values for each band)
*corresponds to expression (7) in Collet article, but possible discrepancy of factor 2; code => $2\theta^2 < (2r)^2 + s^2$ but article => $4\theta^2 < (2r)^2 + s^2$ ***

133 (Sto) ***SeeingTest** parameters are src = 1 (=> single source), band = 'VIS'*
139 (Sto) ***SeeingTest** (=> seeingtest) depends on Einstein radius (bl), **unlensed** source size (rs), seeing, and pixelsize*

280 (FLS) if seeingtest = **Fail** => returns **None** & terminates **makeLens** function

283/203 (FLS) if **MakeModel** = **True** (default) and (seeingtest <> **Fail)** then **makeLens** function runs function **MakeModel** using each band

205/190 (FLS) **MakeModel** runs a function **EvaluateGalaxy** to create **galmodel** (returned as model), which is flux of the *lens* galaxy (in different bands) calculated over all
192/55 (FLS) pixels in the 'postage stamp' (which is a 200 x 200 array <= > 'side' x 'side')
192 (FLS) *can think of 'light**m**' as **m**od of 'light' (negative values are ignored)*
190/205 (FLS) main parameters are *light = gal, mag = ml*
120 (FLS) *gal is an object of class **SBModels.Sersic***
39 (SBP) *which inherits class from **SBProfiles.Sersic**, which has **pylens** function*
23 (SBP) *NoFreeParams = **False** by default*
78 (SBP) *note n = 4 (default) for lens, corresponding to de Vaucouleurs light profile*
122 (FLS) *as per article (p2)*
120 (FLS) *pa = 90 (default value for gal, ie. for a lens galaxy)*
*the **pixeval** function produces a 'distance array', corresponding to the distance of each pixel (in the 'postage stamp') from the coordinates of the galaxy centre; using the Sersic profile, this is then converted into magnitude/flux detectable at each pixel. (The coordinates xl, yl represent the lens galaxy centre, but the profile extends beyond that point into other pixels (requiring the 'postage stamp' grid to include all affected pixels); note that for all **lenses**, certain parameters are the same (eg. centered at 'origin') but different ellipticity q => different pixel configurations*

207/156 (FLS) **MakeModel** runs a function **lensAsource** to create **sourcemodel,** which is the flux of (each) *lensed* source galaxy (in different bands) calculated over all pixels, as well as values for magnification and totallensedsrcmag

158 (FLS) Creates an object **lens** = **PowerLaw** (**massmodel**). This is a sub-class of **_PowerLaw** (**models**) which is a sub-class of **_MassModel** (**models**).

Initialisation of this collects the parameters and their values corresponding to the power law mass-density profile of the lens

159 (FLS) Creates an object **es** = **ExtShear** (**massmodel**). This is a sub-class of **_ExtShear** (**models**) which is a sub-class of **_MassModel** (**models**). Initialisation of this collects the parameters and their values relevant for determining the external shear due to the lens

162-166 (FLS) This imposes a 50x50 grid/'stamp' over the coordinates of the (centre of the)
168 (FLS) *unlensed* source galaxy in order to run the **pixeval** function on its profile; then derives unlensedsrcmodel = **sum of** pixel values for unlensed source
39 (SBP) *uses function **pixeval;** note that parameter n = 1 (default) =>*
79 (SBP) ***pixeval** function returns pixel values as amp.s.reshape (flux)*
77 (SBP) *s is determined from R0, using model (R vs s0) spline*
56 (SBP) *derives 'elliptical distance' for coords x, y; possible discrepancy with distance formula quoted after expression (1) in article ***
58 (SBP) *normalisation => brightness in half-light radius is the half-light radius [SS]*
59-60 (SBP) *sampling of radius; note units are of half-light radius as R in Sersic profile here represents usual 'r'/'r_0' [SS - see also Obs. Cos. p97]*

169 (FLS) *srcnorm = **sum of** pixel values for unlensed source*
170 (FLS) *=> unlensedsrcmodel=unlensedsrcmodel/srcnorm = 1*

172 (FLS) Uses the function **lens_images** (**pylens**) to create an array srcmodel of pixel values corresponding to the **lensed** image of the source.
*parameters include **PowerLaw** and **ExtShear** objects lens, er*
160 (FLS) *since parameter lenses = [lens, er]*

173-174 (FLS) Adjusts srcmodel so that it is an array of pixel values corresponding to the *lensed* source, but each divided by the sum of *unlensed* pixel values, and suppressing negative values
55 (FLS) *array is 200x200 (as self.x, self.y are arguments of pylens.lens_images)*

176 (FLS) magnification calculated as summed srcmodel elements (ie. pixel values) divided by 'normalised' unlensedsrcmodel
but this is same as summed srcmodel, as normalised unlensedsrcmodel=1
(seems OK though as srcmodel is 'already' divided by srcnorm)

179 (FLS) Calculates (unlensedtotalsrcflux =) total **flux of unlensed source** in each band
derived from unlensed source magnitude (ms) and zeropoints

180 (FLS) Calculates (sm =) total **flux of lensed source** in each band
*(returned as sourcemodel in **MakeModel** function) determined by shearing/magnifying total unlensed source flux according to pixel values of lensed source: srcmodel is a 200 x 200 array giving the ratio of summed unlensed pixel values to each lensed pixel value.*

182-185 (FLS) Determines values for magnitudes of lensed source in each band
corresponding to sm flux values

209-212 (FLS) **MakeModel** creates model which is the sum of the *lens* flux values and the *lensed source* flux values (element by element in the array), for each band
model (200x200) = sourcemodel (200x200) + galmodel (200x200)

286-287 (FLS) **makeLens** function runs **stochasticObserving** again if survey strategy = "resolve" (default)
text says need 'to re-run now magnification is known'; this routine tests for LHS of expression (8) in Collett article to select worstacceptedpsf/band
recall floor1 => expression (7), and floor2 => LHS of expression (8)
43-46 (Sto) *magnification is populated now so no **KeyError** as earlier =>*
96-100 (Sto) *PSFfloor values (=> worstacceptedpsf/band) returned are different too*

289-290 (FLS) **makeLens** function runs a function **ObserveLens** and then returns values for galmodel, sourcemodel, model, magnification, totallensedsrcmag

216-225 (FLS) If seeing is non-zero, **ObserveLens** function convolves the image of the **lens** galaxy (galmodel) with the psf and returns both the convolved image of the lens galaxy convolvedgal and an FFT of the psf; then creates convolvedmodel into which convolvedgal (an array) is placed.
217 (FLS) *populates elements for bands list*
220 (FLS) *doPSF is True = > psfFFT is derived in convolution routine*
221 (FLS) *negative values are suppressed*
112-117 (Sto) *seeing based on value derived in **stochasticObserving** function*

229-232 (FLS) **ObserveLens** function derives (if seeing is non-zero) a convolved image of the **source** galaxy (sourcemodel) with the psf and adds this convolved image of the source galaxy convolvedsrc (element by element) to convolvedmodel (for each source).
230 (FLS) *doPSF is **False** => makes use of existing FFT psf so only returns [0] element, namely convolved image, and not FFT psf (cf. line 220)*
231 (FLS) *negative values are suppressed*
235 (FLS) *eg. zeromagcounts,*
237 (FLS) *exposurecorrection,*
241 (FLS) *background,*
245 (FLS) *sigma,*
248 (FLS) *fakelens*
259 (FLS) SN derived from function **SNfunc**
13 (SN) *main parameters are data = convolvedsrc,*
sig = sigma, significancefloor = 0.5 (default)

133/145 (MAll) **From line 133 (MAll):** if this *is* a repeat of the previous survey for that pair i
146 (MAll) then runs function **loadModel** function on S[survey] and;
265 (FLS) **loadModel** function populates objects galmodel, sourcemodel, model (empty), magnification, and totallensedsrcmag (unpacks component by component)

from model; and *parameter model = ideallens in* ***loadModel*** *function*

147 (MAll) Runs the **StochasticObserving** function on S[survey]
266 (FLS) *model is unchanged, but new values for stochastic variables generated*
69-135 (Sto) *for this observation (eg. pfs, sb) => new seeingtest*
148-152 (MAll) if seeingtest (model=> magnified source) = **Fail** then sets pf = **False** and 'continue' => returns to 'for survey in surveys' loop for next survey;
153 (MAll) if seeingtest <> **Fail**, then runs the function **ObserveLens** on S[survey], and then continues to **line 155** (MAll)

134-137 (MAll) **From line 133 (MAll):** If this *is not* a repeat of the previous survey and if
280 (FLS) model type returned from **makeLens** is not **None,** adds 'i' to the numeric part of the lastsurvey value (eg. Euc35, Euc37, Euc60...);
(seeingtest = Fail <=> model = ***None****)*
138-144 (MAll) If seeingtest (model => magnified source) = **Fail**, then sets pf = **False**, rfpf = **False** and 'continue' => returns to 'for survey in surveys' loop for next survey

155 (MAll) **ModelAll** runs **SourceMetaData** (**SignaltoNoise.py**) on S['Euclid'] to return mag (source magnification), msrc (lensed source magnitude per band), SN, bestband and pf, using SN and magnification thresholds a = 20 and b = 3
SourceMetaData *function tests the criteria in expressions (7) - (9)*
155 (MAll) *main parameters are SNcutA = a, magcut = b, SNcutB = [c,d]*
17-18 (MAll) *c = 1000 d = 1000*

57-73 (SN) If seeing = 0, returns resolved = **False**, else applies test corresponding to
59-63 (SN) expression (8) (LHS) in article and if **pass** then returns resolved = **True** otherwise **False**

77 (SN) Uses 'max-iterate-lambda' Python construction to search through SNr dictionary and return both the **key** (band) as **bestband** and the **value** (SN) as **dummy** that are associated with the maximum value (index 1 <=> SN*)*

79 (SN) Sets passfail = **False**

81-83 (SN) Compares SN of element [1] and element [2] of bestband array of values to maximum and minimum (respectively) of SNcutB elements [c, d];
if element [1] > 1000 and element [2] > 1000 => passfail = ***True****;*
not clear why this condition is imposed as it is not reflected in the article, and the limits mean it is unlikely to be met anyway;

91-93 (SN) Applies tests corresponding to expression (8) (both sides) and expression (9) in article; if tests are passed then return passfail = **True**

98-99 (SN) Runs function SeeingTest with seeing[bestband], and if condition not met then returns passfail = **False**
*corresponds to expression (7) in Collet article, but possible discrepancy of factor 2 (see earlier) ***

77 (SN) *bestband corresponds to band with highest SN*

156-173 (MAll) **ModelAll** continues by initialising and populating lenspars parameters SN, bestband, pf (this is passfail), resolved, poptag and seeing
180-181 (MAll) *also populates rfpf ('ringfinder-passfail') and rfsn ('ringfinder-SN') but left **False** and **0** respectively if survey <> 'Euclid';*

220 (MAll) Sets data stored as L.lens to **None** for that background-foreground pair i
*(see **Generate_Lens_Pop** function in **MakeLensPop** module)*
narrative says 'delete used data for memory saving'

222 (MAll) Sets default value accept = **False**

224-225 (MAll) For each survey, sets accept = **True** if pf = **True**

227-237 (MAll) If accept = **True**, then adds 1 to Si (a counter) and copies lenspars values
67-68 (MAll) (for the pair i) into a dictionary SSPL with keyword Si
232-237 (MAll) If Si modulo 1000 = 0, then dumps the data into pickle file as a list comprising elements frac (a number) and SSPL (a dictionary)
239 (MAll) Deletes data stored as **L**.lens for that background-foreground pair i

240 (MAll) Returns to start of 'for i in range' loop

241-244 (MAll) Dumps (residual) data into pickle file (with frac and SSPL as list elements) and displays Si (counter)

****

MakeResults

1 (MRs) Initialisation imports modules including **cPickle**, **pylab**, **glob**, and **pyfits**
3 (MRS) ***pyfits** is not used (or available) and needs to be commented out ***
8-17 (MRs) Sets default parameters for plotting
19/21 (MRs) Sets sourcepops = 'lsst' and experiment = 'Euclid'

26/27 (MRs) *checks if an argument has been passed on the command line, and if so sets it as the experiment name*

29-46 (MRs) Sets up surveystoread as list of surveys (here = 'Euclid')

49-224 (MRs) Runs for survey in surveystoread loop (up to line 224)
51-56 (MRs) *checks survey[-2] character for 'a', 'b', 'c' and replaces with co-add description to return surveyname; but [-2] position is not correct, although surveyname is not used anywhere else anyway **
loop continues as follows:

60-61 (MRs) Opens f = lensparsfile (= lenses_Euclid.txt)
to be used in lines 123-142 (MRs)

64-72 (MRs) Initialises lens and source parameters as dictionaries, including creation of keywords **resolved** and **rfpf** for each (parameter) dictionary:
bl, zs, rs, ms, zl, sigl, ql, mag, weights
weights is 'new' (dictionary) parameter
76/82 (MRs) *note duplication of rs[key]*

73-83 (MRs) Runs for key in ['resolved', 'rfpf'] loop to initialise lens and source parameter as dictionary value arrays (for those keys)
loop => for key = 'resolved' and then for key = 'rfpf';

93-95 (MRs) Sets frac (= 42,000/15,000) and bands (='VIS') parameters for 'Euclid';
127/169 (Sur) *possible discrepancy between values used in **Surveys.py** module (namely, 41,253 and 20,000)*

106 (MRs) Uses Python **glob** function to return pathnames of all Euclid_lsst_Lens_stats* pickle files (* = number of each file) in a list called filelist

108 (MRs) Initialises count chunki
111/113 *used to count (pickle files read) in for chunk in filelist loop*

109 (MRs) Initialises list ilist =[]
119/150 (MRs) *used to store (source-deflector pair) in for i in sspl.keys() loop*

111-201 (MRs) Runs for chunk in filelist loop (**up to line 200**)
chunk = path & filename of each Euclid_lsst_Lens_stats pickle file*

114-117 (MRs) From each **Euclid_lsst_Lens_Stats*** pickle file (chunk), read in first element as fracsky, and the 'remainder' (second element) as dictionary sspl (lenspars data for every source-deflector pair in that pickle file)
note sspl is a dictionary and so too is sspl[i]

119-200 (MRs) Commences for i in sspl.keys() loop (up to line 200)
sspl key is actual number of each source-deflector pair
(i is just for-loop counter)

120-121 (MRs) *if i has already been covered in the loop, then continue => return to beginning of loop for next i*
122-126 (MRs) If i has not been covered in the loop before, checks to see if **seeing** value (in sspl[i] dictionary) exists and if not then continue => return to beginning of loop for next i, otherwise:

127-150 (MRs) Writes source-deflector parameters for pair i from pickle file into **lenses_[survey].txt** file, and appends i to ilist

132/135 (MRs) *note duplication of rl element - it is written **twice** into the*
140/141 (MRs) ***lenses_[survey].txt** file; note also that ms element has been omitted (contrary to narrative in example txt file on GitHub)*****

145-146 (MRs) *rfsn parameter only written if survey is not 'Euclid'*
note position [1] from arrays used for source(s) data (eg. xs[1], zs[1])

151 (MRs) If pf (= passfail) is **False** for that source-deflector pair i, then continue => return to beginning of loop for next i, otherwise:

153-160 (MRs) *checks there exists a value for keyword bestband and, if so, sets it to bb otherwise passes;*

163 (MRs) If the source-deflector pair i can be resolved using the bestband, (<=> resolved[value associated with bestband] = **True**) and if both mag >=3 and SN for the bestband >=20, then appends values for 'new' dictionary parameters bl, zs, rs, ms, zl, sigl, ql, mag, weights with keyword **resolved** using values of parameters b, zs, rs, ms, zl, sigl, ql, mag from sspl

169/116 (MRs) *weights['resolved'] = 1/fract*

64-83 (MRs) *(recall sspl has keywords 'resolved', 'zl', 'zs', 'bestband' etc ; the 'new' parameters are dictionaries with keywords 'resolved' and 'rfpf')*

163 (MRs) *there is no 'else' clause; **False** => proceeds to end of loop at line 200 and then returns to beginning of loop for next i*
165-166 (MRs) *if mag and SN conditions are not met for that source-deflector pair i, then continue => return to beginning of loop for next i*

178-200 (MRs) If the source-deflector pair i meets the galaxy subtraction criteria (<=> rfpf = **True**) and if both rfsn >=20 and RF = **True**, then appends values for 'new' dictionary parameters bl, zs, rs, ms, zl, sigl, ql, mag, weights with keyword **rfpf** using values of parameters b, zs, rs, ms, zl, sigl, ql, mag from sspl

179-180 (MRs) *if rfsn and RF conditions are not met for that source-deflector pair i, then continue => return to beginning of loop for next i; 'Euclid' = > RF = **False** => no values associated with rfpf keyword in new dictionary parameters*

201 (MRs) After ending for i in sspl.keys() loop and (then) for chunk in filelist loop, closes **lenses_[survey].txt** file
203-210 (MRs) *see lines 51-56 above*

212-216 (MRs) Displays results as (a) sum of weights['resolved'] (= 1/fract), and (b) sum of weights ['rfpf'], totalled over all source-deflector pairs, "as lenses found assuming Poisson limited galaxy subtraction in all bands" or "lenses in the g-i difference images" respectively

for Euclid, the first item is simply Si x {(42000/15000).0.1}$^{-1}$
see above 127/169 (Sur) for possible discrepancy
Si is the number of detectable lenses based on data for 0.1 of the whole sky; so for detectable lenses within the actual survey area, Si needs to be scaled by (i) multiplying by 10, and (ii) multiplying by {survey area/whole sky area}

218-220 (MRs) Dumps parameters' dictionary values (associated with keywords resolved and rfpf) into *lists.pkl file
this pkl file will contain values (only) for each of the 9 parameters, for each of the keywords resolved and rfpf

** *denotes my amendment to code on* ***17 September 2017***
*** *denotes amendment* ***reversed*** *on* ***16 November 2017***
**** *denotes my amendment to code on* ***16 November 2017***

Appendix B

Comoving Volume & Deflector Numbers

In order to correct for the issue raised in section 3.2.6, rather than adopt the 'blunt' approach of simply hardwiring the deflector number into the code, I chose to modify the model so that the number of deflectors is derived (by default) according to the Concordance cosmology values: this ensures the correct density function regardless of the cosmological parameters applied elsewhere in the code.

Left uncorrected, the number of deflectors directly affects the number of idealised lenses predicted by the model. Figure B.1 serves to illustrate how this leads to a significant but spurious variation in the number of idealised lenses over a range of values of Ω_m.

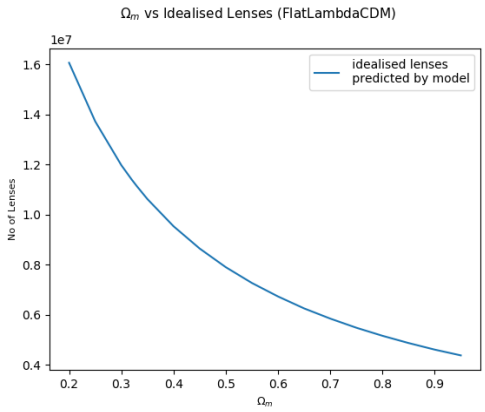

Figure B.1: Idealised Lenses vs Ω_m

The modifications carried out to the code necessitated the introduction of a new class object (*SCDistance*) within the *distances.py* module, as well as amendments to the *MakeLensPop.py* and *PopulationFunctions.py* modules; the source codes of these modules, amended where indicated, are shown below.

```
"""

*** Astrodistances.py ***

REPLACES THE DISTANCES MODULE WITH FlatLambdaCDM ASTROPY COSMOLOGY

A module to compute cosmological distances, including:
    comoving_distance (Dc)
    angular_diameter_distance (Da)
    luminosity_distance (Dl)
    comoving_volume (volume)

"""
import astropy.cosmology
from astropy.cosmology import FlatLambdaCDM
'''
this is the class needed for Astropy distance functions; we could alternatively import
FlatwCDM, rather than wCDM, and not bother with Ode parameter (as FLAT => Ode = 1-Om).
'''

import warnings   # original Collett code and comment
#warnings.warn("Default cosmology is Om=0.3,Ol=0.7,h=0.7,w=-1 and distance units are Mpc!",ImportWarning)

cosmo=(0.20,0.667)  # NOTE COSMOLOGY (FLAT with LAMBDA) **** Tested here with Om = 0.20 ****

class Distance(FlatLambdaCDM):   # INTRODUCTION OF FlatLambdaCDM COSMOLOGY

# Distance is the existing class name used in Collett code; to avoid having to change the name throughout
# the code I have simply kept the classname Distance but have it inherit the FlatLambda/w CDM properties

    def __init__(self):
        FlatLambdaCDM.__init__(self,H0=(cosmo[1]*100),Om0=cosmo[0])
        self.OMEGA_M = cosmo[0]
        print 'Om0 = ', self.OMEGA_M
        print 'w = -1.0'
        print 'h = ',cosmo[1]

       # self.w = -1.
       # self.wpars = None          used in Collett code to accommodate variable w
       # self.w_analytic = False    used in Collett code to accommodate variable w
```

```
41          self.Dc = self.co_distance
42          self.Dt = self.co_t_distance
43          self.Dm = self.co_t_distance
44          self.Da = self.ang_diam_distance
45          self.Dl = self.lum_distance
46          self.dm = self.dist_modulus
47          self.volume = self.co_volume
48
49      # original Collett code has h defined as an attribute, but this is
…   autmatically defined within Astropy
50
51      def co_distance(self,z):
52          return self.comoving_distance(z).value
53
54      def co_t_distance(self,z):
55          return self.comoving_distance(z).value
56
57      def ang_diam_distance(self,z1,z2=0):
58          if z2<z1:
59              z1,z2 = z2,z1
60          return self.angular_diameter_distance_z1z2(z1,z2).value
61
62      def lum_distance(self,z):
63          return self.luminosity_distance(z).value
64
65      def dist_modulus(self,z):
66          if z>0:                                  # needed to avoid runtime
…   'divide by zero' error
67              return self.distmod(z).value
68          else:
69              return 0
70
71      def co_volume(self,z):
72          return self.comoving_volume(z).value
73
74
75  class SCDistance(FlatLambdaCDM):   # INTRODUCTION OF CLASS OBJECT FOR SC
…   FlatLambdaCDM COSMOLOGY - cw
76
77      def __init__(self):
78          FlatLambdaCDM.__init__(self,H0=66.7,Om0=0.324)  # ensures
…   Standard Cosmology used for comoving volume - cw
79
80          self.SCvolume = self.co_volume
81
82      def co_volume(self,z):
83          return self.comoving_volume(z).value
```

```
import distances
from scipy import interpolate
import cPickle,numpy,math
import indexTricks as iT
import pylab as plt
from PopulationFunctions import *

class LensPopulation(LensPopulation_):
    def  __init__(self,zlmax=2,sigfloor=250,D=None,SCD=None,reset=True,

…bands=['F814W_ACS','g_SDSS','r_SDSS','i_SDSS','z_SDSS','Y_UKIRT','VIS']
                  ): #sadface
        self.sigfloor=sigfloor
        self.zlmax=zlmax
        self.bands=bands

        self.beginRedshiftDependentRelation(D,SCD,reset)  # include
…new'SC' comoving volume object
        self.beginLensPopulation(D,SCD,reset)

    def phi(self,sigma,z):
    #you can change this, but remember to reset the splines if you do.
        sigma[sigma==0]+=1e-6
        phi_star=(8*10**-3)*self.D.h**3
        alpha=2.32
        beta=2.67
        sigst=161
        phi=phi_star * \
            ((sigma*1./sigst)**alpha)*\
            numpy.exp(-(sigma*1./sigst)**beta)*beta/\
            math.gamma(alpha*1./beta)/\
            (1.*sigma)

        #phi*=(1+z)**(-2.5)
        self.nozdependence=True

        return phi


class SourcePopulation(SourcePopulation_):   # include new 'SC' comoving
…volume object
    def  __init__(self,D=None,SCD=None,reset=False,

…bands=['F814W_ACS','g_SDSS','r_SDSS','i_SDSS','z_SDSS','Y_UKIRT'],
…population="cosmos"
                  ):
        self.bands=bands
        self.beginRedshiftDependentRelation(D,SCD,reset)
        if population=="cosmos":
```

```
45              self.loadcosmos()
46          elif population=="lsst":
47              self.loadlsst()
48
49
50          #NB all the functions are in the inherited from class.
51
52  #,'VIS'
53
54  #=======================================
55  class LensSample():
56      """
57      Wrapper for all the other objects so you can just call it, and then
    run
58      Generate_Lens_Pop to get a fairly drawn lens population
59      """
60      def  __init__(self,D=None,SCD=None,reset=False,zlmax=2,sigfloor=100,
61
    bands=['F814W_ACS','g_SDSS','r_SDSS','i_SDSS','z_SDSS','Y_UKIRT'],
    sourcepop="lsst"
62                    ):   # cw removed cosmo=[0.3,0.7,0.7] argument as not
    needed with Astropy
63          self.sourcepopulation=sourcepop
64          if D==None:
65              import distances
66              D=distances.Distance() # cw removed 'cosmo=cosmo' argument
    as not needed with Astropy
67              SCD=distances.SCDistance()  # needed for 'SC' comoving
    volume - cw
68
69
    self.L=LensPopulation(reset=reset,sigfloor=sigfloor,zlmax=zlmax,bands=
    bands,D=D,SCD=SCD)
70
71
    self.S=SourcePopulation(reset=reset,bands=bands,D=D,SCD=SCD,population=
    sourcepop)        # include new 'SC' comoving volume object
72
73          self.E=EinsteinRadiusTools(D=D)
74
75      def Lenses_on_sky(self):
76          self.ndeflectors=self.L.Ndeflectors(self.L.zlmax)
77          return self.ndeflectors
78
79      def
    Generate_Lens_Pop(self,N,firstod=1,nsources=1,prunenonlenses=True,save=
    True):
80          import time
81          t0=time.clock()
```

```
82          if prunenonlenses==False: assert N<60000
83
84          self.lens={}
85          self.reallens={}
86          M=N*1
87          l=-1
88          l2=-1
89          while M>0:
90              timeleft="who knows"
91              if M!=N:
92                  tnow=time.clock()
93                  ti=(tnow-t0)/float(N-M)
94                  timeleft=ti*M/60.
95
96
97              print M,timeleft," minutes left"
98              if M>100000:
99                  n=100000
100             else:
101                 n=M*1
102             M-=n
103             zl,sigl,ml,rl,ql=self.L.drawLensPopulation(n)
104
…   zs,ms,xs,ys,qs,ps,rs,mstar,mhalo=self.S.drawSourcePopulation(n*nsources,
…   sourceplaneoverdensity=firstod,returnmasses=True)
105
106             zl1=zl*1
107             sigl1=sigl*1
108             for i in range(nsources-1):
109                 zl=numpy.concatenate((zl,zl1))
110                 sigl=numpy.concatenate((sigl,sigl1))
111
112             b=self.E.sie_rein(sigl,zl,zs)
113             for i in range(n):
114                 l +=1
115                 self.lens[l]={}
116                 if b[i]**2>(xs[i]**2+ys[i]**2):
117                     self.lens[l]["lens?"]=True
118                 else:
119                     self.lens[l]["lens?"]=False
120
121                 self.lens[l]["b"]={}
122                 self.lens[l]["zs"]={}
123                 self.lens[l]["zl"]=zl[i]
124                 self.lens[l]["sigl"]=sigl[i]
125                 for j in range(nsources):
126                     self.lens[l]["zs"][j+1]=zs[i+j*n]
127                     self.lens[l]["b"][j+1] =b[i+j*n]
128
```

```
129                 self.lens[l]["ml"]={}
130                 self.lens[l]["rl"]={}
131                 self.lens[l]["ms"]={}
132 
133                 for band in ml.keys():
134                         self.lens[l]["ml"][band]=ml[band][i]
135                         self.lens[l]["rl"][band]=rl[band][i]
136                 self.lens[l]["ql"]=ql[i]
137 
138                 self.lens[l]["ms"]={}
139                 self.lens[l]["xs"]={}
140                 self.lens[l]["ys"]={}
141                 self.lens[l]["rs"]={}
142                 self.lens[l]["qs"]={}
143                 self.lens[l]["ps"]={}
144                 self.lens[l]["mstar"]={}
145                 self.lens[l]["mhalo"]={}
146 
147                 for j in range(nsources):
148                     self.lens[l]["ms"][j+1]={}
149                     for band in ml.keys():
150                         self.lens[l]["ms"][j+1][band]=ms[band][i+j*n]
151                     self.lens[l]["zs"][j+1]=zs[i+j*n]
152                     self.lens[l]["b"][j+1] =b[i+j*n]
153                     self.lens[l]["xs"][j+1]=xs[i+j*n]
154                     self.lens[l]["ys"][j+1]=ys[i+j*n]
155                     self.lens[l]["rs"][j+1]=rs[i+j*n]
156                     self.lens[l]["qs"][j+1]=qs[i+j*n]
157                     self.lens[l]["ps"][j+1]=ps[i+j*n]
158                     self.lens[l]["mhalo"][j+1]=mhalo[i+j*n]  #CW -
… corrected transposed elements for halo & star
159                     self.lens[l]["mstar"][j+1]=mstar[i+j*n]
160 
161 
162                 if self.lens[l]["lens?"]:
163                     if prunenonlenses:
164                         l2+=1
165 
166                         self.reallens[l2]=self.lens[l].copy()
167 
168                         del self.lens
169                         self.lens={}
170 
171                         if l2%1000==0:
172                             print l2
173 
174                         if (l2+1)%10000==0:
175                           if save:
176 
```

```
176… fn="idealisedlenses/lenspopulation_%s_%i.pkl"%(self.sourcepopulation,l2-
  … 10000+1)
177                             print fn
178                             f=open(fn,'wb')
179                             cPickle.dump(self.reallens,f,2)
180                             f.close()
181                             del self.reallens
182                             self.reallens={}
183
184                 elif prunenonlenses:
185                     del self.lens
186                     self.lens={}
187         if save:
188
  … fn="idealisedlenses/lenspopulation_%s_residual_%i.pkl"%(self.
  … sourcepopulation,l2)
189             print l2,fn
190             f=open(fn,'wb')
191             cPickle.dump(self.reallens,f,2)
192             f.close()
193
194         if prunenonlenses==False:
195           if save:
196
  … f=open("idealisedlenses/nonlenspopulation_%s.pkl"%self.sourcepopulation,
  … 'wb')
197             cPickle.dump(self.lens,f,2)
198             f.close()
199             print len(self.lens.keys())
200
201         self.lens=self.reallens
202
203     def LoadLensPop(self,j=0,sourcepopulation="lsst"):
204
  … f=open("idealisedlenses/lenspopulation_%s_%i.pkl"%(sourcepopulation,j),'
  … rb')
205         self.lens=cPickle.load(f)
206         f.close()
207
208
209     def Pick_a_lens(self,i=None,dspl=False,tspl=False):
210         if i ==None:
211             numpy.random.randint(0,self.n)
212
213         self.rli={}
214         self.mli={}
215         self.msi={}
216         self.msi2={}
217         self.msi3={}
```

```
218
219         for band in self.L.bands:
220             self.rli[band]=self.rl[band][i]
221             self.mli[band]=self.ml[band][i]
222         for band in self.S.bands:
223             self.msi[band]=self.ms[band][i]
224             if dspl or tspl:
225                 self.msi2[band]=self.ms2[band][i]
226                 if tspl:self.msi3[band]=self.ms3[band][i]
227
228
…   preselection=self.apply_preselection(self.mli["i_SDSS"],self.zl[i])
229         if dspl==False and tspl==False:
230             return
…   [self.mli,self.rli,self.ql[i],self.bl[i]],[self.msi,self.xs[i],self.yl[i
…   ],self.qs[i],self.ps[i],self.rs[i]],[self.zl[i],self.zs[i]],preselection
231         elif tspl==False:
232             return
…   [self.mli,self.rli,self.ql[i],self.bl[i]],[self.msi,self.xs[i],self.yl[i
…   ],self.qs[i],self.ps[i],self.rs[i]],[self.bl2[i],self.msi2,self.xs2[i],
…   self.yl2[i],self.qs2[i],self.ps2[i],self.rs2[i]],[self.zl[i],self.zs[i],
…   self.zs2[i],self.sigl[i],self.Mvs[i],self.r_phys[i]],preselection
233
234         else:
235             return [self.mli,self.rli,self.ql[i],self.bl[i]],
…   [self.msi,self.xs[i],self.yl[i],self.qs[i],self.ps[i],self.rs[i]],
…   [self.bl2[i],self.msi2,self.xs2[i],self.yl2[i],self.qs2[i],self.ps2[i],
…   self.rs2[i]],
…   [self.bl3[i],self.msi3,self.xs3[i],self.yl3[i],self.qs3[i],self.ps3[i],
…   self.rs3[i]],
…   [self.zl[i],self.zs[i],self.zs2[i],self.zs3[i],self.sigl[i],self.Mvs[i],
…   self.r_phys[i]],     preselection
236
237
238     def apply_preselection(self,imag,z):
239         if imag<15: return False
240         if imag>23:return False
241         if z<0.05: return False
242         return True
243
244 if __name__ == "__main__":
245     import distances
246     fsky=1
247     D=distances.Distance()
248     SCD=distances.SCDistance()    # creates class of object needed for
…   comoving volume with Omega_m = 0.324 and h = 0.667 ('SC') - cw
249     Lpop=LensPopulation(reset=True,sigfloor=100,zlmax=2,D=D,SCD=SCD)
250     Ndeflectors=Lpop.Ndeflectors(2,zmin=0,fsky=1)
251     L=LensSample(reset=False,sigfloor=100,sourcepop="lsst")   # cw
```

```
251… removed cosmo argument as not needed for Astropy
252
…  L.Generate_Lens_Pop(int(Ndeflectors),firstod=1,nsources=1,prunenonlenses
…  =True)
253
```

```
import distances
from scipy import interpolate
import cPickle,numpy,math
import indexTricks as iT

#=============================================================================
==============

class RedshiftDependentRelation():
    def __init__(self,D=None,SCD=None,reset=False): # cw removed
cosmo=[0.3,0.7,0.3] arg as not needed for Astropy
        self.beginRedshiftDependentRelation(D,SCD,reset=reset) # cw
removed cosmo=cosmo arg as not needed for Astropy

    def beginRedshiftDependentRelation(self,D,SCD,reset,zmax=10):  # cw
removed cosmo=[0.3,0.7,0.3] arg as not needed for Astropy
        self.zmax=zmax
        self.zbins,self.dz=numpy.linspace(0,self.zmax,401,retstep=True)

self.z2bins,self.dz2=numpy.linspace(0,self.zmax,201,retstep=True)
        if D==None:
            import distances
            D=distances.Distance()
            SCD=distances.SCDistance()
        self.D=D
        self.SCD=SCD

        if reset!=True:
            try:
            #load useful redshift splines
                splinedump=open("redshiftsplines.pkl","rb")

self.Da_spline,self.Dmod_spline,self.volume_spline,self.SCvolume_spline,
self.Da_bispline=cPickle.load(splinedump)
            except IOError or EOFError:
                self.redshiftfunctions()
        else:
            self.redshiftfunctions()

    def redshiftfunctions(self):
        D=self.D
        SCD=self.SCD
        zbins=self.zbins
        z2bins=self.z2bins
        Dabins=zbins*0.0
        Dmodbins=zbins*0.0
        Da2bins=numpy.zeros((z2bins.size,z2bins.size))
        volumebins=zbins*0.0
        SCvolumebins=zbins*0.0    # new variable for SC comoving volume
```

```
42… - cw
43          for i in range(zbins.size):
44              Dabins[i]=D.Da(zbins[i])                #   OK for astropy
45              Dmodbins[i]=D.dm(zbins[i])              #   OK for astropy
46              volumebins[i]=D.volume(zbins[i])        #   OK for astropy
47              SCvolumebins[i]=SCD.SCvolume(zbins[i])     # new variable
… for SC comoving volume - cw
48          for i in range(z2bins.size):
49              for j in range(z2bins.size):
50                  if j>i:
51                      Da2bins[i,j]=D.Da(z2bins[i],z2bins[j])    #   OK for
… astropy
52
53          self.Da_spline=interpolate.splrep(zbins,Dabins)
54          self.Dmod_spline=interpolate.splrep(zbins,Dmodbins)
55
56          self.volume_spline=interpolate.splrep(zbins,volumebins)
57          self.SCvolume_spline=interpolate.splrep(zbins,SCvolumebins)  #
… new variable for SC comoving volume - cw
58
59          z2d=iT.coords((z2bins.size,z2bins.size))*self.dz2
60
… self.Da_bispline=interpolate.RectBivariateSpline(z2bins,z2bins,Da2bins)
61
62          #pickle the splines
63          splinedump=open("redshiftsplines.pkl","wb")
64
… cPickle.dump([self.Da_spline,self.Dmod_spline,self.volume_spline,self.
… SCvolume_spline,self.Da_bispline],splinedump,2)  # includes new SC
… comoving volume - cw
65
66      def Volume(self,z1,z2=None):    #CAUTION: This has a capital V in
… volume; not used for astropy
67          if z2==None:
68              return self.splev(z1,self.volume_spline)
69          else:
70              z1,z2=self.biassert(z1,z2)
71              return
… self.splev(z2,self.volume_spline)-self.splev(z1,self.volume_spline)
72
73      def SCVolume(self,z1,z2=None):    # Needed for comoving volume with
… Omega_m = 0.324 and h = 0.667 ('SC')  - cw
74          if z2==None:
75              return self.splev(z1,self.SCvolume_spline)
76          else:
77              z1,z2=self.biassert(z1,z2)
78              return
… self.splev(z2,self.SCvolume_spline)-self.splev(z1,self.SCvolume_spline)
79
```

```
80
81      def Da(self,z1,z2=None,units="Mpc"):
82          if units=="kpc":
83              corfrac=1000        #CHECK THIS FUNCTION - doesn't conflict
 … with Da in Astropy.Distances?
84          elif units=="Mpc":
85              corfrac=1
86          else:
87              print "don't know those units yet"
88          if z2==None:
89              return self.splev(z1,self.Da_spline)*corfrac
90          else:
91              z1,z2=self.biassert(z1,z2)
92              return self.Da_bispline.ev(z1,z2)*corfrac
93
94      def Dmod(self,z):
95          return self.splev(z,self.Dmod_spline)
96
97      def splev(self,x,f_of_x_as_spline):
98          return interpolate.splev(x,f_of_x_as_spline)
99
100     def bisplev(self,x,y,f_ofxy_as_bispline):
101         return interpolate.bisplev(x,y,f_ofxy_as_bispline)
102
103     def biassert(self,z1,z2):
104             try: len(z1)
105             except TypeError:z1=[z1]
106             try: len(z2)
107             except TypeError:z2=[z2]
108             if len(z1)==1 and len(z2)!=1:z1=numpy.ones(len(z2))*z1[0]
109             if len(z2)==1 and len(z1)!=1:z2=numpy.ones(len(z1))*z2[0]
110             assert len(z1)==len(z2),"get it together"
111             return z1,z2
112
113 #=============================================================================
 …==============
114
115
116 class EinsteinRadiusTools(RedshiftDependentRelation):
117     def  __init__(self,D=None,SCD=None,reset=False):
118         self.beginRedshiftDependentRelation(D,SCD,reset)
119         self.c=299792
120
121     def sie_sig(self,rein,zl,zs):
122         self.c=299792
123         ds=self.Da(zs)
124         dls=self.Da(zl,zs)
125         sig=(rein*(ds*self.c**2)/(206265*4*math.pi*dls))**0.5
126         return sig
```

```
127     def sie_rein(self,sig,zl,zs):
128         self.c=299792
129         ds=self.Da(zs)
130         dls=self.Da(zl,zs)
131         rein=sig**2*((ds*self.c**2)/(206265*4*math.pi*dls))**-1
132         rein[rein<0]=0
133         return rein
134 
135 
136 #=============================================================================
    ==============
137 class Population(RedshiftDependentRelation):
138     def  __init__(self):
139         pass
140 
141     def draw_apparent_magnitude(self,M,z,band=None,colours=None):
142         if band!=None:
143             colours=self.colour(z,band)
144         if colours==None:
145             colours=0
146             print "warning no k-correction"
147         Dmods=self.Dmod(z)
148         ml = M - colours + Dmods
149         return ml
150 
151     def draw_apparent_size(self,r_phys,z):
152         rl = r_phys/(self.Da(z,units="kpc"))
153         rl *= 206264
154         return rl
155 
156 #=============================================================================
    ==============
157 
158 class LensPopulation_(Population):
159     def  __init__(self,zlmax=2,sigfloor=100,D=None,SCD=None,reset=True,
160 
    bands=['F814W_ACS','g_SDSS','r_SDSS','i_SDSS','z_SDSS','Y_UKIRT','VIS']
161                   ): #sadface     # cw removed cosmo=[0.3,0.7,0.3] arg
    as not needed for Astropy
162         self.sigfloor=sigfloor
163         self.zlmax=zlmax
164         self.bands=bands
165 
166         self.beginRedshiftDependentRelation(D,SCD,reset)  # includes new
    comoving volume ('SC') object - cw
167         self.beginLensPopulation(D,SCD,reset)
168 
169 
170     def beginLensPopulation(self,D,SCD,reset):   # includes new comoving
```

```
170… volume ('SC') object - cw
171         reset=True
172         if reset!=True:
173             try:
174             #load Lens-population splines
175                 splinedump=open("lenspopsplines.pkl","rb")
176 
…   self.cdfdNdzasspline,self.cdfdsigdzasspline,self.dNdzspline,self.
…   SCdNdzspline,self.zlbins,zlmax,sigfloor,self.colourspline,bands=cPickle.
…   load(splinedump) # include new 'SC' dNdzspline
177             except IOError or EOFError or ValueError:
178                 self.lenspopfunctions()
179             #check sigfloor and zlmax are same as requested
180             if zlmax!=self.zlmax or self.sigfloor!=sigfloor:
181                 self.lenspopfunctions()
182             #check all the necessary colours are included
183             redocolours=False
184             for band in self.bands:
185                 if band not in bands:redocolours=True
186             if redocolours:
187                 self.Colourspline()
188                 self.lensPopSplineDump()
189         else:
190             self.lenspopfunctions()
191 
192     def lenspopfunctions(self):
193         self.Psigzspline()
194         self.Colourspline()
195         self.lensPopSplineDump()
196 
197     def Psigzspline(self):
198         #"""
199         #drawing from a 2d pdf is a pain; should probably make this into
…   its own module
200 
…   self.zlbins,self.dzl=numpy.linspace(0,self.zlmax,201,retstep=True)
201         sigmas=numpy.linspace(self.sigfloor,400,401)
202         self.sigbins=sigmas
203         dNdz=self.zlbins*0
204         SCdNdz=self.zlbins*0
205         Csiggivenz=numpy.zeros((sigmas.size,self.zlbins.size))
206         CDFbins=numpy.linspace(0,1,1001)
207         siggivenCz=numpy.zeros((CDFbins.size,self.zlbins.size))
208         for i in range(len(self.zlbins)):
209             z=self.zlbins[i]
210             dphidsiggivenz=self.phi(sigmas,z)
211             phisigspline=interpolate.splrep(sigmas,dphidsiggivenz)
212             tot=interpolate.splint(self.sigfloor,500,phisigspline)
213 
```

```
213…  Csiggivenz[:,i]=numpy.cumsum(dphidsiggivenz)/numpy.sum(dphidsiggivenz)
214               Csiggivenzspline=interpolate.splrep(Csiggivenz[:,i],sigmas)
215               siggivenCz[:,i]=interpolate.splev(CDFbins,Csiggivenzspline)
216               if z!=0:
217
  …   dNdz[i]=tot*(self.Volume(z)-self.Volume(z-self.dzl))/self.dzl
218
  …   SCdNdz[i]=tot*(self.SCVolume(z)-self.SCVolume(z-self.dzl))/self.dzl
219
220           Nofzcdf=numpy.cumsum(dNdz)/numpy.sum(dNdz)
221           SCNofzcdf=numpy.cumsum(SCdNdz)/numpy.sum(SCdNdz)   # new
  …   variable using 'SC' comoving volume - cw
222           #import pylab as plt
223           #plt.plot(self.zlbins,Nofzcdf)
224           #plt.show()
225           #exit()
226           self.cdfdNdzasspline=interpolate.splrep(Nofzcdf,self.zlbins)
227           self.SCcdfdNdzasspline=interpolate.splrep(SCNofzcdf,self.zlbins)
228
229           self.dNdzspline=interpolate.splrep(self.zlbins,dNdz)
230           self.SCdNdzspline=interpolate.splrep(self.zlbins,SCdNdz)
231
232           N=interpolate.splint(0,self.zlmax,self.dNdzspline)
233           SCN=interpolate.splint(0,self.zlmax,self.SCdNdzspline)
234
235           self.cdfdsigdzasspline=interpolate.RectBivariateSpline(\
236               CDFbins,self.zlbins,siggivenCz)
237
238           dphidsiggivenz0=self.phi(sigmas,sigmas*0)
239           cdfdNdsigz0=dphidsiggivenz0.cumsum()/dphidsiggivenz0.sum()
240           self.cdfdNdsigz0asspline=interpolate.splrep(cdfdNdsigz0,sigmas)
241
242
243           #"""
244           #phi is redshift independant.
245
246
247
248       def Colourspline(self):
249           from stellarpop import tools
250           sed = tools.getSED('BC_Z=1.0_age=10.00gyr')
251           #different SEDs don't change things much
252
253           rband=tools.filterfromfile('r_SDSS')
254           z=self.zlbins
255           self.colourspline={}
256           for band in self.bands:
257             if band!="VIS":
258               c=z*0
```

```
259             Cband=tools.filterfromfile(band)
260             for i in range(len(z)):
261                 c[i] = - (tools.ABFM(Cband,sed,z[i]) -
…  tools.ABFM(rband,sed,0))
262             self.colourspline[band]=interpolate.splrep(z,c)
263
264
265     def lensPopSplineDump(self):
266         splinedump=open("lenspopsplines.pkl","wb")
267
…  cPickle.dump([self.cdfdNdzasspline,self.cdfdNdsigz0asspline,self.
…  cdfdsigdzasspline,self.dNdzspline,self.SCdNdzspline,self.zlbins,self.
…  zlmax,self.sigfloor,self.colourspline,self.bands],splinedump,2)
268
269     def draw_z(self,N):
270         return
…  interpolate.splev(numpy.random.random(N),self.cdfdNdzasspline)
271
272     def draw_sigma(self,z):
273         try: len(z)
274         except TypeError:z=[z]
275         if self.nozdependence:
276             sigs
…  =interpolate.splev(numpy.random.random(len(z)),self.cdfdNdsigz0asspline)
277             return sigs
278         else:
279             print "Warning: drawing from 2dpdf is low accuracy"
280             return
…  self.cdfdsigdzasspline.ev(numpy.random.random(len(z)),z)
281
282     def draw_zsig(self,N):
283         z=self.draw_z(N)
284         sig=self.draw_sigma(z)
285         return z,sig
286
287     def EarlyTypeRelations(self,sigma,z=None,scatter=True,band=None):#z
…  dependence not encoded currently
288         #Hyde and Bernardi, M = r band absolute magnitude.
289         V=numpy.log10(sigma)
290         Mr=(-0.37+(0.37**2-(4*(0.006)*(2.97+V)))**0.5)/(2*0.006)
291         if scatter:
292             Mr+=numpy.random.randn(len(Mr))*(0.15/2.4)
293
294         #R=4.72+0.63*Mr+0.02*Mr**2 #rest-frame R_band size.
295         R=2.46-2.79*V+0.84*V**2
296         if scatter:
297             R+=numpy.random.randn(len(R))*0.11
298
299         #convert to observed r band size;
```

```
300         r_phys = 10**R
301 
302         return Mr,r_phys
303 
304     def colour(self,z,band):
305         return interpolate.splev(z,self.colourspline[band])
306 
307     def Ndeflectors(self,z,zmin=0,fsky=1):
308         if zmin>z:
309             z,zmin=zmin,z
310         N=interpolate.splint(zmin,z,self.dNdzspline)
311         SCN=interpolate.splint(zmin,z,self.SCdNdzspline)
312         #N*=fsky            # need to choose
313         SCN*=fsky          # between N and SCN
314         #return N            # in these
315         return SCN         # lines
316 
317     def phi(self,sigma,z):
318         sigma[sigma==0]+=1e-6
319         phi_star=(8*10**-3)*self.D.h**3
320         alpha=2.32
321         beta=2.67
322         sigst=161
323         phi=phi_star * \
324             ((sigma*1./sigst)**alpha)*\
325             numpy.exp(-(sigma*1./sigst)**beta)*beta/\
326             math.gamma(alpha*1./beta)/\
327             (1.*sigma)
328 
329         phi*=(1+z)**(-2.5)
330         return phi
331 
332     def draw_flattening(self,sigma,z=None):
333         x=sigma
334         y=0.378-0.000572*x          # CW - reinstated minus coefficient
… as typo in article rather than in code
335         e=numpy.random.rayleigh(y)
336         q=1-e
337         #dont like ultraflattened masses:
338         while len(q[q<0.2])>0 or len(q[q>1])>0:
339             q[q<0.2]=1-numpy.random.rayleigh(y[q<0.2])
340             q[q>1]=1-numpy.random.rayleigh(y[q>1])
341         return q
342 
343     def drawLensPopulation(self,number):
344         self.zl,self.sigl=self.draw_zsig(number)
345         self.ql=self.draw_flattening(self.sigl)
346 
… self.Mr,self.r_phys_nocol=self.EarlyTypeRelations(self.sigl,self.zl,
```

```
346… scatter=True)
347          self.ml={}
348          self.rl={}
349          self.r_phys={}
350          for band in self.bands:
351              self.r_phys[band]=self.r_phys_nocol#could add a colorfunc
…  here
352              if band !="VIS":
353
…  self.ml[band]=self.draw_apparent_magnitude(self.Mr,self.zl,band)
354              else: pass
355
…  self.rl[band]=self.draw_apparent_size(self.r_phys[band],self.zl)
356          return self.zl,self.sigl,self.ml,self.rl,self.ql
357
358 #=================================================================
…  ==============
359
360 class SourcePopulation_(Population):
361      def  __init__(self,D=None,SCD=None,reset=False,
362
…  bands=['F814W_ACS','g_SDSS','r_SDSS','i_SDSS','z_SDSS','Y_UKIRT','VIS'],
…  population="cosmos"
363                    ):  # cw removed cosmo=[0.3,0.7,0.3] arg as not needed
…  for Astropy
364          self.bands=bands
365
366          self.beginRedshiftDependentRelation(D,SCD,reset)  # include 'SC'
…  comoving volume object
367
368          if population=="cosmos":
369              self.loadcosmos()
370          elif population=="lsst":
371              self.loadlsst()
372
373      def loadcosmos(self):
374          self.population="cosmos"
375
376          try:
377              #load pickledcosmos
378              cosmosdump=open("cosmosdata.pkl","rb")
379              cosmosphotozs=cPickle.load(cosmosdump)
380          except IOError or EOFError:
381              import re
382
…  photozs=open('../Forecaster/cosmos_zphot_mag25.tbl','r').readlines()[10:
…  ]
383              splinedump=open("cosmosdata.pkl","wb")
384              cols=len(re.split(r"\s+",photozs[0])[1:-1])
```

```
385             rows=len(photozs)
386             cosmosphotozs=numpy.empty((cols,rows))
387             for i in range(len(photozs)):
388                 line=photozs[i]
389                 l=numpy.array(re.split(r"\s+",line)[1:-1])
390                 l[l=='null']=999
391                 cosmosphotozs[:,i]=l
392             cosmosphotozs=cosmosphotozs.astype(numpy.float)
393             raz=cosmosphotozs[2,:]
394             decz=cosmosphotozs[3,:]
395             zc=cosmosphotozs[6,:]
396             cosmosphotozs=cosmosphotozs[:,((zc<10)&(zc>0))]
397             cPickle.dump(cosmosphotozs,splinedump,2)
398 
399         self.zc=cosmosphotozs[6,:]
400 
401         self.m={}
402         index={}
403         index["g_SDSS"]=23 #lets pretend sdss_g=cfht_g etc
404         index["r_SDSS"]=24
405         index["i_SDSS"]=25
406         index["z_SDSS"]=26
407         index["Y_UKIRT"]=27 #pretend Y_DES=ic whatever ic is...
408         index["F814W_ACS"]=25 # But we'll make do with F814==i
409 
410         for band in self.bands:
411           if band!="VIS":
412             self.m[band]=cosmosphotozs[index[band],:]
413 
… self.m["VIS"]=(self.m["r_SDSS"]+self.m["i_SDSS"]+self.m["z_SDSS"])/3
414 
415         self.Mv=cosmosphotozs[-1,:]
416 
417         self.mstar=cosmosphotozs[-1,:]*0.
418         self.mhalo=cosmosphotozs[-1,:]*0.
419 
420     def loadlsst(self):
421         self.population="lsst"
422         import cPickle
423 
424         f=open('lsst.1sqdegree_catalog2.pkl','rb')
425         print "new lsst catalogue"
426         data=cPickle.load(f)
427         f.close()
428 
429         self.zc=data[:,2]
430         self.m={}
431         #print data[:,0].max()-data[:,0].min()
432         #print data[:,1].max()-data[:,1].min()
```

```
433
434         self.m["g_SDSS"]=data[:,3]
435         self.m["r_SDSS"]=data[:,4]
436         self.m["i_SDSS"]=data[:,5]
437         self.m["z_SDSS"]=data[:,6]
438         self.m["F814W_ACS"]=data[:,5] # we'll make do with F814==i
439         self.m["Y_UKIRT"]=data[:,6]*99 #there is no Y band data atm
440         self.mstar=data[:,12]
441         self.mhalo=data[:,13]
442
…  self.m["VIS"]=(self.m["r_SDSS"]+self.m["i_SDSS"]+self.m["z_SDSS"])/3
443         self.Mv=data[:,7]
444
445     def RofMz(self,M,z,scatter=True,band=None):#band independent so far
446     #{mosleh et al}, {Huang, Ferguson et al.}, Newton SLACS XI.
447         r_phys=((M/-19.5)**-0.22)*((1.+z)/5.)**(-1.2)
448         # is the same as
449         R=-(M+18.)/4.
450         r_phys=(10**R)*((1.+z)/1.6)**(-1.2)
451
452         if scatter!=False:
453             if scatter==True:scatter=0.35 #dex
454             self.scattered=10**(numpy.random.randn(len(r_phys))*scatter)
455             r_phys*=self.scattered
456
457         return r_phys
458
459
460     def draw_flattening(self,N):
461         y=numpy.ones(N*1.5)*0.3
462         e=numpy.random.rayleigh(y)
463         q=1-e
464         q=q[q>0.2]
465         q=q[:N]
466
467         return q
468
469     def
…  drawSourcePopulation(self,number,sourceplaneoverdensity=10,returnmasses=
…  False):
470         source_index=numpy.random.randint(0,len(self.zc),number*3)
471         #source_index=source_index[((self.zc[source_index]<10) &
…  (self.zc[source_index]>0.05))]
472         source_index=source_index[:number]
473         self.zs=self.zc[source_index]
474         self.Mvs=self.Mv[source_index]
475         self.ms={}
476         for band in self.bands:
477             if band !="VIS":
```

```
478                 self.ms[band]=self.m[band][source_index]
479             else:
480
…   self.ms[band]=(self.m["r_SDSS"][source_index]+self.m["i_SDSS"][
…   source_index]+self.m["z_SDSS"][source_index])/3.
481
482         self.r_phys=self.RofMz(self.Mvs,self.zs,scatter=True)
483         self.rs=self.draw_apparent_size(self.r_phys,self.zs)
484         self.qs=self.draw_flattening(number)
485
486         self.ps=numpy.random.random_sample(number )*180
487
488         #cosmos has a source density of ~0.015 per square arcsecond
489         if self.population=="cosmos":
490             fac=(0.015)**-0.5
491             a=fac*(sourceplaneoverdensity)**-.5
492         #lsst sim has a source density of ~0.06 per square arcsecond
493         elif self.population=="lsst":
494             fac=(0.06)**-0.5
495             a=fac*(sourceplaneoverdensity)**-.5
496
497         else:
498             pass
499
500         self.xs=(numpy.random.random_sample(number)-0.5)*a
501         self.ys=(numpy.random.random_sample(number)-0.5)*a
502
503         if returnmasses:
504             self.mstar_src=self.mstar[source_index]
505             self.mhalo_src=self.mhalo[source_index]
506             return
…   self.zs,self.ms,self.xs,self.ys,self.qs,self.ps,self.rs,self.mstar_src,
…   self.mhalo_src
507
508         return self.zs,self.ms,self.xs,self.ys,self.qs,self.ps,self.rs
509
510
511 class AnalyticSourcePopulation_(SourcePopulation_):
512     def  __init__(self,D=None,reset=False,
513
…   bands=['F814W_ACS','g_SDSS','r_SDSS','i_SDSS','z_SDSS','Y_UKIRT']
514                   ):  # cw removed cosmo=[0.3,0.7,0.3] arg as not needed
…   for Astropy
515         self.bands=bands
516         self.beginRedshiftDependentRelation(D,reset)
517         print "not written!"
518
519
520
```

```
521 if __name__=="__main__":
522     #RedshiftDependentRelation(reset=True)
523 
524     #L=LensPopulation_(reset=True,sigfloor=100)
525 
526     S=SourcePopulation_(reset=False,population="cosmos")
527     S2=SourcePopulation_(reset=False,population="lsst")
528 
529 
530     print
…   numpy.median(S.Mv[S.m["i_SDSS"]<25])-numpy.median(S2.Mv[S2.m["i_SDSS"]<
…   25])
531     print
…   len(S.Mv[S.m["i_SDSS"]<25])*1./(len(S2.Mv[S2.m["i_SDSS"]<25])*100)
532     print len(S.Mv)/(60.**2)/2.
533     print len(S2.Mv[S2.m["i_SDSS"]<25])/(0.2**2)/(60.**2)
534     print len(S2.Mv)/(0.2**2)/(60.**2)
535 
536     #print EarlyTypeRelations(self,100,z=None,scatter=True,band=None)
537 
538 
539 
540 
541 
542 
543 
544 
545 
546 
547 
548 
549 
550 
551 
552 
553 
554 
555 
556 
557 
558 
559 
560 
561 
562 
563 
564 
565 
566 
```

```
567
568
569
570
571
572
573
574
575
576
577
578
579
```

Appendix C

Source Galaxy Light Profile

In this appendix, we look further at the discrepancy highlighted in section 3.2.8, namely the method by which the effective radius is derived for the light profile of a (source) galaxy.

According to expression (5) in Collett (2015), the effective radius R_e is given by:

$$log_{10}(R_e/kpc) = (M_v/-19.5)^{-0.22} \times ((1+z)/5)^{-1.2} + a\ scatter\ factor$$

The routine corresponding to this expression in the code lies within the definition of the *RofMz* function, which commences at line 421 in the PFs module.

There are several discrepancies between the code and Collett (2015), and indeed within the code itself. In particular, the *RofMz* function includes the following lines:-

1. line 423 (PFs)

$$r_{phys} = ((M/-19.5)^{-0.22)} \times ((1+z)/5)^{-1.2}$$

   which the narrative says is the same as:

2. line 425 (PFs)

$$R = -(M+18)/4$$

3. line 426 (PFs))

$$R_{phys} = (10^R) \times ((1+z)/1.6)^{-1.2}$$

The code in line 423 is subsequently overwritten by the code in lines 425 and 426, so does not directly impact on the results of the model. However, the narrative implies that the code in line 425 and 426 is *equivalent* to that of line 423.

We consider firstly whether the code in line 423 accurately reflects expression (5) in Collett (2015) and, if it does not, whether the discrepancy lies within the code or within the text of the article; we then further consider the effect of overwriting line 423 with lines 425 and 426. In doing so, we note the use of r_{phys} subsequent to line 423 for deriving the 'apparent size' rs of a galaxy (line 425), which implies that R_e in Collett (2015) corresponds to r_{phys} in the code.

An inconsistency here is immediately apparent: the expression on the RHS of the code in line 423 is equivalent to the RHS of expression (5), but the LHS of expression (5) is given as the log_{10} of the effective radius rather than the effective radius itself.

From Mosleh et al. (2012) (e.g. see section 4.2 and the caption to fig. 5), the effective radius (r_e) is modelled as a function of redshift z by:

$$r_e \propto (1+z)^{-1.2 \pm 0.11}$$

which is consistent with the expression in Wyithe & Loeb (2011), namely:

$$r_e \propto (1+z)^{-1.2 \pm 0.17}$$

Neither of these are consistent however with expression (5) in Collett (2015), which relates instead the log_{10} of the effective radius as a power function of $(1+z)$. On the other hand, they are consistent with the *redshift* dependence given by the code in line 423 (up to a constant).

Now, in addition to redshift, expression (5) includes a dependence of the effective radius on the galaxy's absolute Magnitude M.

From Huang et al. (2013) (p18), the effective radius of a galaxy can be expressed as a function of luminosity L as:

$$r_e \propto L^{\beta}$$

where β takes values of $0.22^{+0.058}_{-0.056}$ and $0.25^{+0.15}_{-0.14}$ (depending on the sample)[1].

Since luminosity and magnitude are related by the formula:

$$-2.5log_{10}L + k = M$$

for k constant, then we can rewrite the size-luminosity function as:

$$log_{10}r_e \propto \beta log_{10}L$$

$$\Rightarrow log_{10}r_e \propto \beta\frac{M}{-2.5} + k'$$

for k' iconstant.

According to this study therefore, the log_{10} of the effective radius is *linearly* related to the magnitude of the galaxy. This is inconsistent with expression (5) in Collett (2015), which shows log_{10} of the effective radius to be a *power function* of the magnitude. It also appears to be inconsistent with the code in line 423, which shows the effective radius (as opposed to its log_{10}) as a power function of the magnitude.

[1]Note **a** to Table 4 in Collett (2015) states $r_e \propto L^{-\beta}$: this is assumed to be a typographical error.

Assuming therefore that the effective radius can be written as a function of luminosity and redshift together, then from the above we have (ignoring error ranges):-

- for the redshift

$$r_e \propto (1+z)^{-1.20}$$

- for the magnitude

$$log_{10} r_e \propto 0.22 \frac{M}{-2.5}$$

$$\Rightarrow r_e \propto 10^{\frac{0.22M}{-2.5}}$$

The implication of this derivation is that both expression (5) in Collett (2015) and the code in line 423 seem to contain misprints. In both cases, there appears to have been a confusion of $log_{10} r_e$ with r_e.

We now turn to the consequences of the code in lines 425 and 426, which effectively overwrite the code in line 423.

We note firstly that lines 425 and 426 do not readily flow from line 423; it is difficult to see how (if at all) they can be equivalent to it, as stated in the narrative. However, if in line 425 we define R as $log_{10} r\_phys$ then the dependence of the effective radius on the magnitude is linear (and negative) - which is consistent with the derivation above. This in turn means the dependence of r_phys on magnitude (as a power of ten) given in line 426 is plausible.

We further note that in line 426, the dependence of the effective radius on the redshift is also consistent (up to a constant) with the derivations above.

In summary therefore, whereas expression (5) in Collett (2015) and the code in line 423 are inconsistent - both with each other and with the studies cited - the expression represented by lines 425 and 426 does at least appear defensible. Since the code in those two lines is responsible for the results obtained by the model, then we may conclude no amendments are necessary other than perhaps to comment out or delete the redundant line 423.

Appendix D

Implementation of Astropy

D.1 Modified Distances module

The following is an example of the source code used to replace the original *distances.py* module with a module designed to call the functions available from the **astropy** package instead.

The functions from the original *distances.py* module alongside the **astropy** equivalents used to replace them are shown in Table D.1.

```
"""


REPLACES THE DISTANCES MODULE

A module to compute cosmological distances, including:
    comoving_distance (Dc)
    angular_diameter_distance (Da)
    luminosity_distance (Dl)
    comoving_volume (volume)

"""

import astropy.cosmology
from astropy.cosmology import FlatLambdaCDM   # this is the class needed for Astropy distance functions

import warnings
warnings.warn("Default cosmology is Om=0.3,Ol=0.7,h=0.7,w=-1 and distance units are Mpc!",ImportWarning)

cosmo=(0.3,0.7,0.7)  # NOTE COSMOLOGICAL PARAMETERS AS ABOVE!

class Distance(FlatLambdaCDM):

    # Distance is the existing class name used in Collett code; to avoid having to change the name throughout
    # the code I have simply kept the classname Distance but have it inherit the FlatLambdaCDM properties

    def __init__(self):
        FlatLambdaCDM.__init__(self,H0=(cosmo[2]*100),Om0=cosmo[0])
        self.OMEGA_M = cosmo[0]
        self.OMEGA_L = cosmo[1]
        self.w = -1.
       # self.wpars = None          used in Collett code to accommodate variable w
       # self.w_analytic = False    used in Collett code to accommodate variable w
        self.Dc = self.co_distance
        self.Dt = self.co_t_distance
        self.Dm = self.co_t_distance
        self.Da = self.ang_diam_distance
        self.Dl = self.lum_distance
        self.dm = self.dist_modulus
        self.volume = self.co_volume

     # original Collett code has h defined as an attribute, but this is autmatically defined within Astropy
```

```
43
44      def co_distance(self,z):
45          return self.comoving_distance(z).value
46
47      def co_t_distance(self,z):
48          return self.comoving_distance(z).value
49
50      def ang_diam_distance(self,z1,z2=0):
51          if z2<z1:
52              z1,z2 = z2,z1
53          return self.angular_diameter_distance_z1z2(z1,z2).value
54
55      def lum_distance(self,z):
56          return self.luminosity_distance(z).value
57
58      def dist_modulus(self,z):
59          if z>0:                                  # needed to avoid runtime
… 'divide by zero' error
60              return self.distmod(z).value
61          else:
62              return 0
63
64      def co_volume(self,z):
65          return self.comoving_volume(z).value
```

Table D.1: Distance module functions & **astropy** equivalents

Distance module function	Line nos.	**astropy** equivalent	Comment
comoving_distance (Dc)	23/50	comoving_distance	-
comoving_transverse_distance (Dt)	24/73	comoving_transverse_distance	For default (flat) cosmology, $Dc = Dt$; see lines 74-85
comoving_transverse_distance (Dm))	25/73	comoving_transverse_distance	Dm definition is identical to Dt, indicating its alternative potential definition as 'proper motion distance'
angular_diameter_distance (Da)	26/87	angular_diameter_distance_z1z2	Applies to two objects at redshifts z1 and z2; redshift z2=0 by default (code will swap z1 and z2 to ensure $z2 > z1$)
luminosity_distance (Dl)	27/92	luminosity_distance	-
distance_modulus (dm)	26/109	dist_mod	An 'if' statement has to be included here to return zero in the event the argument $z = 0$, to avoid a 'divide by zero on log_{10}' warning
comoving_volume (volume)	29/95	comoving_volume	-
age	42	not required	Returns value for age of the universe; not used within the model.
rho_crit	105	not required.	Returns value for ρ_{crit}; not used within the model.

D.2 Wide & Deep Field Surveys with Astropy

The data from the Euclid Wide & Deep Field surveys obtained before *and* after the modification for **astropy** are summarised in Table D.2 and Table D.3, where in both cases the results *prior* to **astropy** are shown in italics. The results serve as a 'sense check' on the modification and, as can be verified by inspection, are reassuringly consistent for each of the two surveys.

Table D.2: Wide Field Predictions

Parameter	Mean	Median	Variance
Lens redshift	0.71 *0.71*	0.64 *0.64*	0.13 *0.14*
Source redshift	1.93 *1.93*	1.82 *1.82*	0.75 *0.75*
Einstein radius (arcsec)	0.72 *0.72*	0.64 *0.63*	0.16 *0.16*
Velocity dispersion (km/s)	220 *220*	218 *218*	2383 *2432*
Source magnitude	25.45 *25.45*	25.46 *25.47*	1.40 *1.41*
Magnification	7.22 *7.21*	5.15 *5.16*	37.4 *36.4*

Table D.3: Deep Field Predictions

Parameter	Mean	Median	Variance
Lens redshift	0.76 *0.76*	0.69 *0.69*	0.15 *0.15*
Source redshift	2.25 *2.25*	2.17 *2.17*	0.81 *0.81*
Einstein radius (arcsec)	0.75 *0.75*	0.66 *0.66*	0.17 *0.18*
Velocity dispersion (km/s)	219 *219*	217 *217*	2445 *2443*
Source magnitude	26.69 *26.68*	26.89 *26.89*	0.96 *0.98*
Magnification	5.30 *5.32*	4.26 *4.27*	10.90 *11.1*

Appendix E

Data Analysis

E.1 Plotting Histograms

Idealised Lenses - Initial Data (Uncorrected Deflector Volume)

This section displays histogram comparisons corresponding to the results from a population comprising *idealised* lenses (as opposed to *detectable* lenses), and *prior* to corrections for the deflector volume: see Figures E.1 to E.4.

An example of the source code written to produce plots such as these is also given. The script in the example was designed to allow simple variations throughout the course of the project, in order to read and analyse data pertaining to different sets of lensing data (both idealised and detectable, as well as 'volume-adjusted'); the results of those are shown in the main text.

```
'''
This program is designed to read a set of specified idealised lens
pickle files and plot
the values for the source redshift (zs), source less lens redshift
(zs-zl), the Einstein Radius (b), and
the velocity dispersion (sigl) of all the lens systems contained in
those files. It also provides
a (histogram) comparison to a set of Concordance (LambdaCDM) idealised
lenses.
.

CFW
8/6/2018

'''
import time
import os
import numpy as np
import matplotlib.pyplot as plt
import pickle
from scipy import stats

# initialise the TEST COSMOLOGY arrays for numpy to analyse
zlarray=[]
zsarray=[]
barray=[]
siglarray=[]
siglsqdarray=[]

# read in data from a sample of 5 TEST Cosmology pickle files of 10,000
items each
for j in range(1,6):
    picklepathandname='/Users/charles/Documents/Collet/Code runs/Extreme
Om/LensPopA_NewVol_LCDM_Om_329/idealisedlenses/lenspopulation_lsst_%
i00000.pkl'%(j)
 #   picklepathandname='/Users/charles/Documents/Collet/Code
runs/All_Cosmo_PLANCK_Astropy/LensPopA_LambdaCDM/idealisedlenses/
lenspopulation_lsst_%i000000.pkl'%(j)  identity test!!
    pickle_in=open(picklepathandname,'rb')
    picklename = os.path.basename(picklepathandname)    # this will
return the filename (without the pathname)
    picklenumber = int(filter(str.isdigit,picklename))   # this will
return just the number in that filename
    if j==1:
        print
        print 'TEST COSMOLOGY DATA PREPARATION ...'
        print 'Reading pkl file: ', picklepathandname
    else:
        print 'Reading pkl file: ', picklename
```

```
39
40      idealised_lens=pickle.load(pickle_in)
41
42  # read in values from the j'th TEST cosmology pickle file
43      x=picklenumber   # automatically sets according to start of pickle
…   file rows.
44      for i in range(x,x+len(idealised_lens)):
45          sigl= idealised_lens[i]['sigl']
46          zs=idealised_lens[i]['zs'][1]
47          zl=idealised_lens[i]['zl']
48          b=idealised_lens[i]['b'][1]
49          lenstrue=idealised_lens[i]['lens?']
50          if lenstrue==False:
51              print 'non-lens found!!'  # just to warn if there is a
…   non-lens item here!
52
53          zsarray=np.append(zsarray,zs)  # use for numpy statistic
…   calculations
54          zlarray=np.append(zlarray,zl)
55          barray=np.append(barray,b)
56          siglarray=np.append(siglarray,sigl)
57  testcountrecs=len(zsarray)  # tells us how many records read in from pkl
…   files
58
59  # initialise the CONCORDANCE cosmology arrays for numpy to analyse
60  conzsarray=[]
61  conzlarray=[]
62  conbarray=[]
63  consiglarray=[]
64  conb_siglsqdarray=[]
65
66  # read in data from  a sample of 5 CONCORDANCE cosmology pickle files of
…   10,000 items each
67  for k in range (1,6):
68      concordpathandname='/Users/charles/Documents/Collet/Code
…   runs/All_Cosmo_PLANCK_Astropy/LensPopA_LambdaCDM/idealisedlenses/
…   lenspopulation_lsst_%i000000.pkl'%(k)
69      concord_in=open(concordpathandname,'rb')
70      concordname = os.path.basename(concordpathandname)
71      concordnumber = int(filter(str.isdigit,concordname))
72      concord_lens=pickle.load(concord_in)
73
74      if k ==1:
75          print
76          print "CONCORDANCE COSMOLOGY DATA PREPARATION ..."
77          print 'Reading pkl file: ', concordpathandname
78      else:
79          print 'Reading pkl file: ', concordname
80
```

```
# read in values from the k'th CONCORDANCE cosmology pickle file
    y=concordnumber  # automatically sets according to start of pickle file rows
    for m in range(y,y+len(concord_lens)):
        consigl= concord_lens[m]['sigl']
        conzs=concord_lens[m]['zs'][1]
        conzl=concord_lens[m]['zl']
        conb=concord_lens[m]['b'][1]
        conlenstrue=concord_lens[m]['lens?']
        if conlenstrue==False:
            print 'non-lens found!!'  # just to warn if there is a non-lens item here!

        conzsarray=np.append(conzsarray,conzs)  # use for numpy statistic calculations
        conzlarray=np.append(conzlarray,conzl)
        conbarray=np.append(conbarray,conb)
        consiglarray=np.append(consiglarray,consigl)
concountrecs=len(conzsarray)   # tells us how many records read in from pkl files

#create new variable as Einstein radius divided by square of velocity dispertion

siglsqdarray=siglarray**2
b_siglsqdarray=barray/siglsqdarray

consiglsqdarray=consiglarray**2
conb_siglsqdarray = conbarray/consiglsqdarray


zbinsize=0.25    # set bin sizes
b_siglsqdbinsize=0.000001

maxzs=10.25  # set max limits
maxzl=2.25
maxb_siglsqd=.00004



print '========================'
print
print 'TEST COSMOLOGY STATISTICS'
print
print 'Mean value for SOURCE REDSHIFT (zs) = ', np.mean(zsarray), '; standard deviation = ', np.std(zsarray)
print 'Range from: ', np.min(zsarray), 'to ', np.max(zsarray),': Concordance mean = ', np.mean(conzsarray)
print
```

```
123 print 'Mean value for LENS REDSHIFT (zl) = ', np.mean(zlarray), ';
  … standard deviation = ', np.std(zlarray)
124 print 'Range from: ', np.min(zlarray), 'to ', np.max(zlarray),':
  … Concordance mean = ', np.mean(conzlarray)
125 print
126 print 'Mean value for EINSTEIN RADIUS (b) = ', np.mean(barray), ';
  … standard deviation = ', np.std(barray)
127 print 'Range from: ', np.min(barray), 'to ', np.max(barray),':
  … Concordance mean = ', np.mean(conbarray)
128 print
129 print 'Mean value for VELOCITY DISPERSION (sigl) = ',
  … np.mean(siglarray), '; standard deviation = ', np.std(siglarray)
130 print 'Range from: ', np.min(siglarray), 'to ', np.max(siglarray),':
  … Concordance mean = ', np.mean(consiglarray)
131 print
132 print 'Mean value for b/sigl_sqd = ', np.mean(b_siglsqdarray), ';
  … standard deviation = ', np.std(b_siglsqdarray)
133 print 'Range from: ', np.min(b_siglsqdarray), 'to ',
  … np.max(b_siglsqdarray),': Concordance mean = ',
  … np.mean(conb_siglsqdarray)
134
135 print
136 print '**********'
137
138 samplesize=len(zlarray)
139
140 titlefilename=os.path.relpath(picklepathandname,'/Users/charles/
  … Documents/Collet/Code runs')
141 head1,tail1 = os.path.split(titlefilename)
142 head2,tail2 = os.path.split(head1)
143 head3,tail3 = os.path.split(head2)
144
145 #plot the top of two graphs
146 plt.suptitle(str(tail3),size=10)
147 plt.subplot(2,1,1)   #use this line to create top of two plots on same
  … page
148 plt.hist(zlarray, bins=50,
  … histtype='step',normed=True,label="lens",color='blue')
149 plt.hist(zsarray-zlarray, bins=50,
  … histtype='step',normed=True,label="source-lens",color='orange')
150 plt.hist(conzlarray, bins=50,
  … histtype='step',normed=True,color='black',label='Concordance',linestyle=
  … 'dotted')
151 plt.hist(conzsarray-conzlarray, bins=50,
  … histtype='step',normed=True,color='black',linestyle='dotted')
152 plt.axis([0,5,0,1.2])
153 plt.xlabel("Redshift",size = 8)
154 plt.ylabel("Probability Density",size=8)
155 #plt.title("Idealised sample - lens vs source redshift",size=8)
```

```
156 plt.text(1.19,0.9,'Idealised - lens & source redshift', size = 9)  # use
  … plt.text to place the text
157 plt.legend()
158
159 # plot the bottom of two graphs
160 plt.subplot(2,1,2)   #use this line to create bottom of two plots on
  … same page
161 plt.hist(b_siglsqdarray,bins=50,
  … histtype='step',normed=True,color='green')
162 plt.hist(conb_siglsqdarray,bins=50,
  … histtype='step',normed=True,color='black',label="Concordance",linestyle=
  … 'dotted')
163 plt.axis([0.0,0.00003,0,80000.0])
164 plt.xlabel("Einstein radius/velocity dispersion_squared",size = 8)
165 plt.ylabel("Probability Density", size=8)
166 plt.text(0.0000125,23000,"Idealised - $\Theta_E / \sigma^2$",size = 9)
167 plt.text(255,0.006,"(normalised)",size = 9)
168 plt.subplots_adjust(top=0.95,bottom=0.08,hspace=0.25,wspace=0.2)
169 plt.legend()
170 #plt.tight_layout()
171 plt.show()
172
```

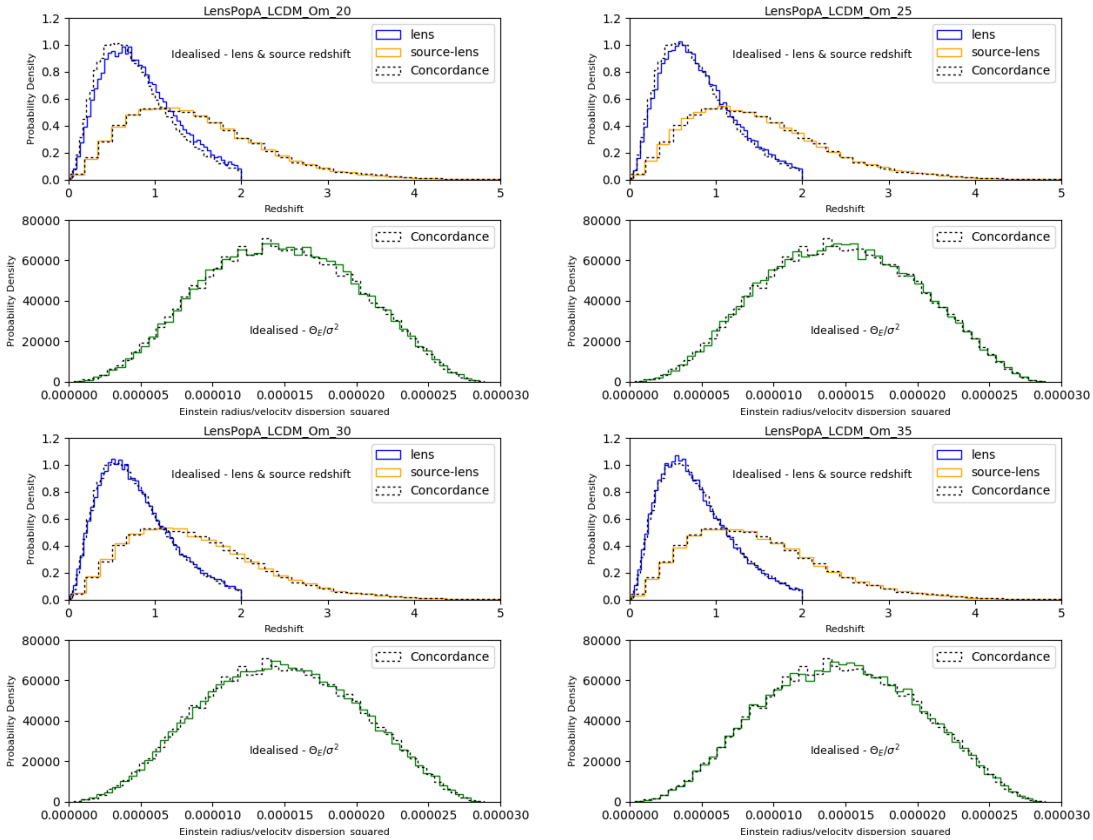

Figure E.1: Properties of idealised lenses: results obtained for a range of values for the density parameter Ω_m prior to any correction for the deflector volume.

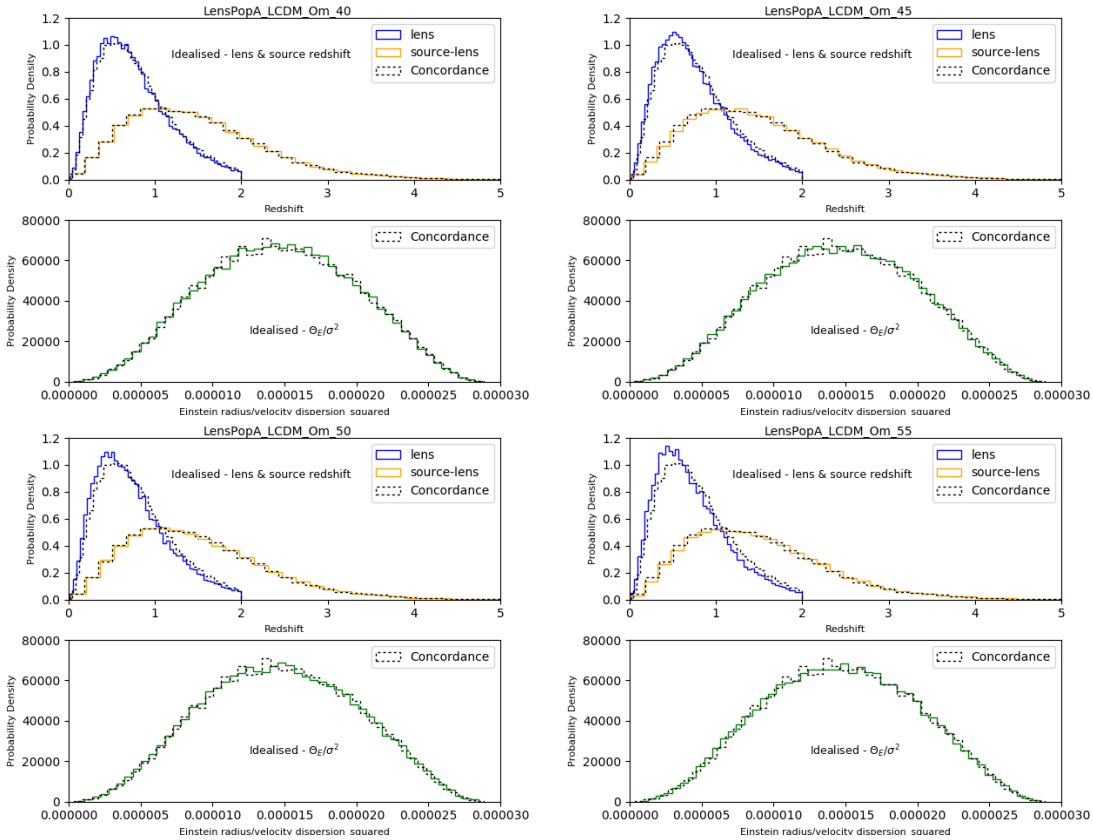

Figure E.2: Continued from Figure E.1. Properties of idealised lenses: results obtained for a range of values for the density parameter Ω_m prior to any correction for the deflector volume.

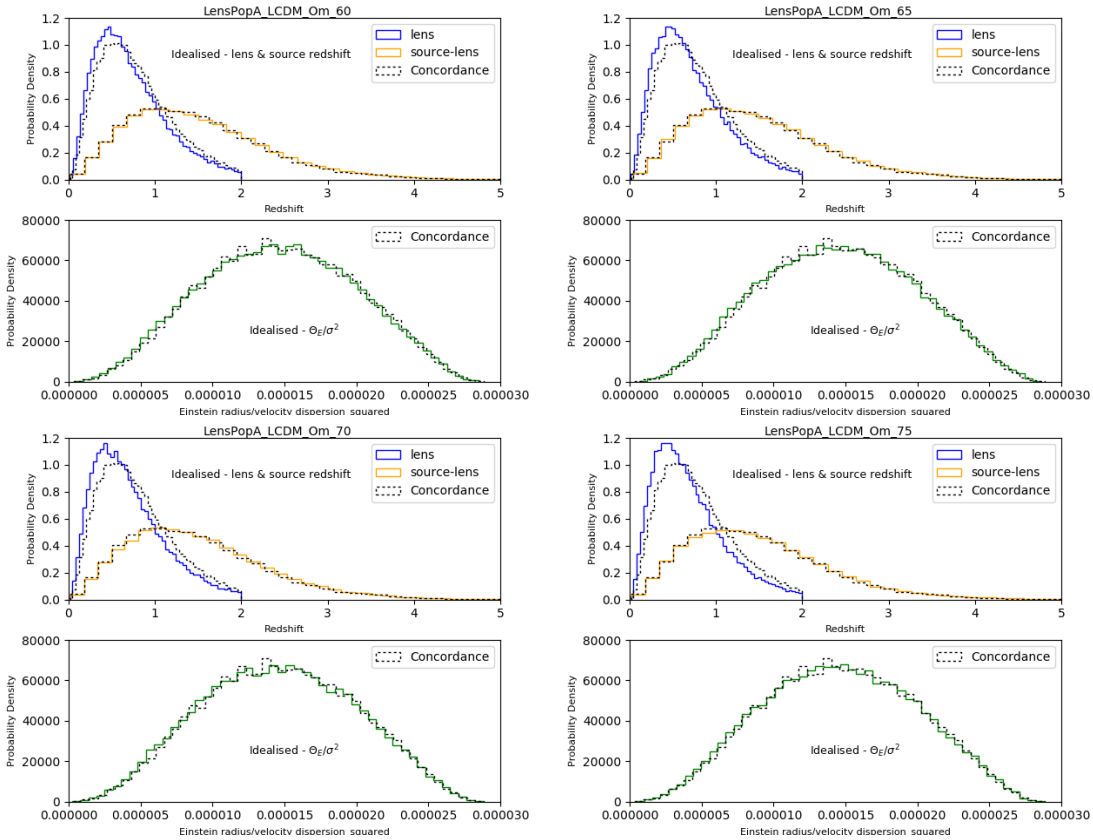

Figure E.3: Continued from Figure E.1. Properties of idealised lenses: results obtained for a range of values for the density parameter Ω_m prior to any correction for the deflector volume.

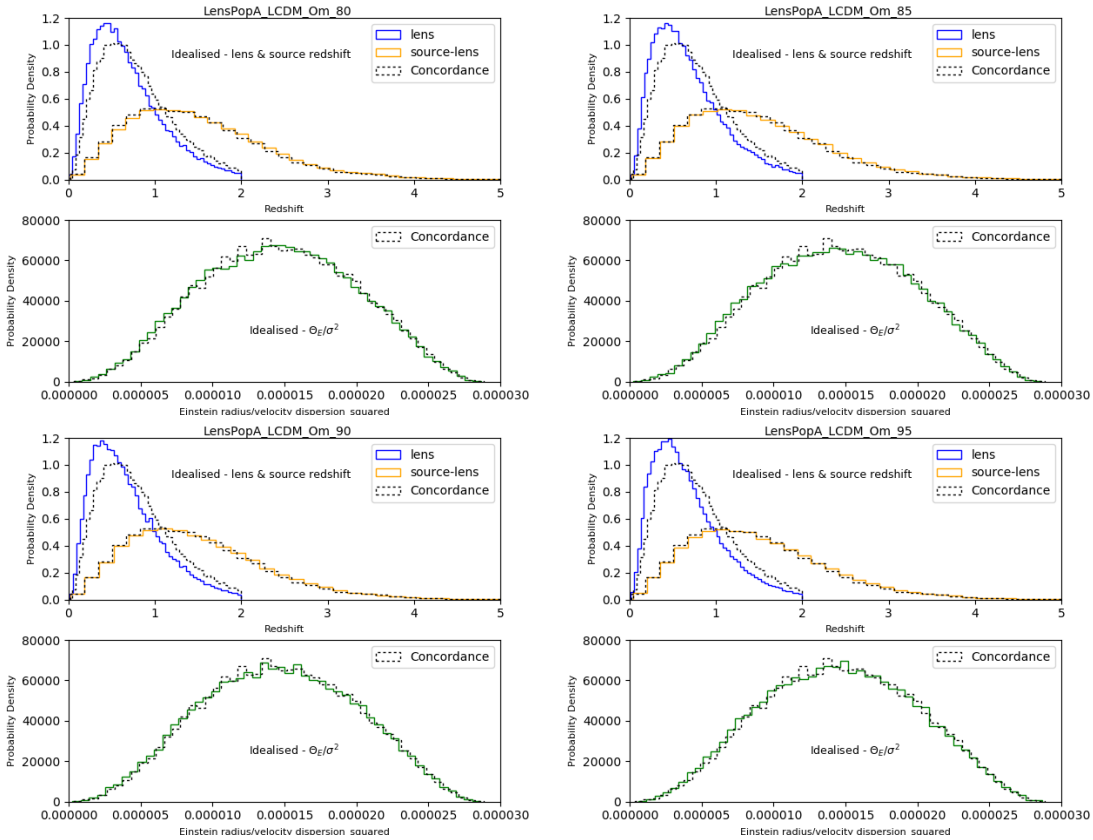

Figure E.4: Continued from Figure E.1. Properties of idealised lenses: results obtained for a range of values for the density parameter Ω_m prior to any correction for the deflector volume.

E.2 Comparing Histograms

In section 5.2, I discuss using a Chi-Square analysis in order to quantitatively compare histograms of differing cosmologies, and I further refer to helpful articles by Bityukov et al. (2013*a*,*b*). In the example source code presented below, it may be useful to note that the variable Si ('significance of deviation') referred to in those articles can be related to χ^2 by means of the following formula:

$$S_i = \frac{n_{i1} - n_{i2}}{\sqrt{\sigma_{ni1}^2 + \sigma_{ni2}^2}}$$

where n_{ik} is the number of events in bin i for histogram k and σ_{ik} is the corresponding standard deviation. For M observations, the root mean square (RMS) of S_i is given by:

$$RMS = \sqrt{\frac{\Sigma_i^M (S_i - \overline{S})^2}{M}}$$

$$= \sqrt{\frac{\chi^2}{M} - \overline{S}^2}$$

where

$$\chi^2 = \Sigma_{i=1}^M S_i^2$$

Two further points to make are that, in the comparisons, bins that are empty across *both* sets of data are excluded from the calculations, and we also assume that observations in each bin are Poisson distributed (ie. the mean is equal to the variance).

An example of the source code written to carry out the histogram comparisons in this project is shown below.

```
'''
For two cosmologies f (Standard Cosmology) and g (Test Cosmology), this code is designed to display:-
 (1) the (zl, zs, b/sigl^2) bin(s) with the highest count for each of the cosmologies
 (2) the (zl, zs, b/sigl^2) bin(s) that displays the greatest difference between the two cosmologies
 (3) the value of the (zl, zs, b/sigl^2) bins(s) with the (absolute) highest 'Significance of Deviation Si' (Bityukov) for the two cosmologies
 (4) the mean value of Si (averaged over all bins) for the two cosmologies
 (5) the RMS of Si for the two cosmologies
 (6) the mean Chi Square value (= RMS**2 + meanSi**2)
 (7) Z-test & p-value for Chi Square.

NOTE: values are read in from txt file as float items to avoid them being read in as strings

CAUTION: This report is designed to ignore bins that are empty across BOTH cosmologies.

CFW
5/6/2018
'''
from scipy import stats
import numpy as np
import time
f=open("/Users/charles/Documents/Collet/Code runs/All_Cosmo_PLANCK_Astropy/LensPopA_LambdaCDM/lenses_Euclid.txt","r")
# ensure correct path entered here!!
#g=open("/Users/charles/Documents/Collet/Code runs/All_Cosmo_PLANCK_Astropy/LensPopA_LambdaCDM/lenses_Euclid.txt","r")
# use this to test for identical distributions
g=open("/Users/charles/Documents/Collet/Code runs/All_Cosmo_PLANCK_Astropy/LensPopA_wCDM/lenses_Euclid.txt","r")   # ensure correct path entered here!!
print
print 'FILES READ'
print '=========='
print 'Standard Cosmology: ', f.name
print
print 'Test Cosmology: ',g.name

# the following prepares the results for the f (later 'sc') file.

paramlist=[]    #initialises what will become a list of lists; that is, a list of lens pair parameters (each lens being a list)
zl=[]           #initialises parameter lists
```

```
35 zs=[]
36 b=[]
37 sigl=[]
38 #mag=[]
39
40 samplesize=5    # use this for testing with limited sample size only
41 scalefactor = 15000/4200   # check this applies to BOTH txt files
42
43 '''
44 CAUTION: Histogram counts need to be SCALED UP for actual predicted
   counts
45 (eg. multiply counts by 15000/4200 for EUCLID wide field surveys)
46 '''
47
48 lenscounterf=0
49
50 for line in f:
51     if line.startswith('#'):    # ignores any commented out rows (eg.
   headings) in the text file
52         continue
53     elements=line.split()    # reads in txt parameters treating each
   line as a list (of parameters for a lens pair)
54     paramlist.append(elements)    # builds up a list of 'lists' - ie. a
   list containing a list of parameters for each pair
55
56 f.close()
57
58 for i in range (len(paramlist)):    # loops through each 'list' in the
   list of 'lists' and builds up a list of individual parameters
59 #for i in range (samplesize):    # use this for testing with limited f
   sample size
60     lenscounterf+=1
61 #    print 'lens pair no: ', lenscounterf (useful for testing 'f' sample
   size)
62
63     zl.append(float(paramlist[i][0]))  # produces a list of zl values
64     zs.append(float(paramlist[i][1]))  # produces a list of zs values
65     b.append(float(paramlist[i][2]))   # produces a list of b (Einstein
   radius) values
66     sigl.append(float(paramlist[i][3])) # produces a list of sigl values
67     #mag.append(float(paramlist[i][13])) # produces a list of mag
   (magnification) values
68
69 # the following prepares the results for the g (later "tc") file, using
   the same procedure as above
70
71 paramlistg=[]
72 zlg=[]
73 zsg=[]
```

```
bg=[]
siglg=[]
#magg=[]

lenscounterg=0

for line in g:
    if line.startswith('#'):
        continue
    elementsg=line.split()
    paramlistg.append(elementsg)

g.close()

for i in range (len(paramlistg)):
#for i in range (samplesize):     #use this line for testing with limited g sample size
    lenscounterg+=1
#    print 'lens pair no: ', lenscounterg (useful for testing 'g' sample size)

    zlg.append(float(paramlistg[i][0]))
    zsg.append(float(paramlistg[i][1]))
    bg.append(float(paramlistg[i][2]))
    siglg.append(float(paramlistg[i][3]))
   # magg.append(float(paramlistg[i][13]))


siglsqd=np.array(sigl)**2     # create new variables from b and sigl, and from bg and siglg
barray=np.array(b)
b_siglsqd=barray/siglsqd

siglsqdg=np.array(siglg)**2
bgarray=np.array(bg)
b_siglsqdg=bgarray/siglsqdg

'''
Now we have to 'couple' the 1-D sets of data to produce an array of multi-D data;
for example, {xi} and {yi} become {xi, yi} or with {zi} we get {xi, yi, zi}
(an alternative method to the one used here would be to invoke the Python ZIP function).
'''
combiarrays=np.array([zl,zs,b_siglsqd],float)  # this creates an array but it is the wrong shape
reshapedarray=combiarrays.transpose()  # the transpose is needed to give us the correct shape
```

```
115
116 # print reshapedarray   # use this to see (zl, zs, b_siglsqd)
    … 'coordinates' of each source-lens pair from f file
117
118 zlbinsize=0.25    # set bin sizes
119 zsbinsize=0.25
120 b_siglsqdbinsize=0.000001
121
122 maxzs=10.25  # set max limits
123 maxzl=2.25
124 maxb_siglsqd=.00004
125
126
127 # create a histogram with bin edges given for zl, zs and b_siglsqd
    … respectively; note ranges are 0->2.25, 0->6.25 and 0.0->0.00004
    … respectively
128 H, edges =
    … np.histogramdd(reshapedarray,(np.arange(0.0,maxzl,zlbinsize),np.arange(0
    … .0,maxzs,zsbinsize),np.arange(0.0,maxb_siglsqd,b_siglsqdbinsize))) #
    … check compatibility with Hg below
129
130 #print 'H :', H
131 #print 'scaling factor: ', scalefactor
132 #print 'bins for zl, zs, b_siglsqd respectively: ', edges   # use this
    … to  print out bins
133 print
134
135 maxHbins=np.where(H==H.max())    # return index (indices) for zs, zl,
    … b_siglsqd respectively of corresponding bin (bins)
136 #print 'f - coordinates (zl, zs, b_siglsqd) of bin with highest count:
    … ', maxHbins
137
138 scaledHmax=scalefactor*H.max()   # need to scale up count in bin to get
    … survey prediction for that bin
139
140 print 'STANDARD COSMOLOGY ("SC")'
141 print '=========================='
142 print 'predicted no. of lenses', scalefactor*lenscounterf
143 print 'lens sample output size:',lenscounterf
144 print 'maximum count
    … (unscaled):',H.max(),'(',round(100*H.max()/lenscounterf,2),'% of
    … sample)'
145 #print 'maximum bin count (scaled)', scaledHmax  # shows scaled up count
    … in bin with highest count
146 print 'occurs in bin: '   #this is unscaled count
147
148 maxHzlbins=maxHbins[0]   # returns position of zs bin(s) corresponding
    … to highest count
149 maxHzsbins=maxHbins[1]   # returns position of zl bin(s) corresponding
```

```
149… to highest count
150 maxHb_siglsqdbins=maxHbins[2]    # returns position of b_siglsqd bin(s)
  … corresponding to highest count
151
152 #print 'maxcount zl bin no: ', maxHzlbins
153 #print 'maxcount zs bin no: ' , maxHzsbins
154 #print 'maxcount b_siglsqd bin no: ', maxHb_siglsqdbins
155
156 print 'zl: ',  (edges[0][(maxHzlbins)]),' - ',
  … (edges[0][(maxHzlbins)])+zlbinsize    # returns zs bin(s) corresponding
  … to highest count
157 print 'zs: ',  (edges[1][(maxHzsbins)]),' - ',
  … (edges[1][(maxHzsbins)])+zsbinsize    # returns zl bin(s) corresponding
  … to highest count
158 print 'b_siglsqd : ',  (edges[2][(maxHb_siglsqdbins)]),' - ',
  … (edges[2][(maxHb_siglsqdbins)])+b_siglsqdbinsize   # returns b bin(s)
  … corresponding to highest count
159 # NOTE: these are not individual zl, zs, b_siglsqd maxima: they are the
  … zl, zs and b_siglsqd values that give the most common (zl, zs,
  … b_siglsqd) combination
160
161 combiarraysg=np.array([zlg,zsg,b_siglsqdg],float)
162 reshapedarrayg=combiarraysg.transpose()
163
164 #print reshapedarrayg   # use this to see (zl, zs, b_siglsqd)
  … 'coordinates' of each source-lens pair from g file
165
166 Hg, edgesg =
  … np.histogramdd(reshapedarrayg,(np.arange(0.0,maxzl,zlbinsize),np.arange(
  … 0.0,maxzs,zsbinsize),np.arange(0.0,maxb_siglsqd,b_siglsqdbinsize)))
167 #print 'Hg :', Hg
168
169 maxHgbins=np.where(Hg==Hg.max())
170 #print 'g - coordinates (zl, zs, b_siglsqd) of bin with highest count:
  … ', maxHgbins
171
172 print
173 scaledHgmax=scalefactor*Hg.max()
174
175 print 'TEST COSMOLOGY ("TC")'
176 print '====================='
177 print 'predicted number of lenses:', scalefactor*lenscounterg
178 print 'lens sample output size:',lenscounterg
179 print 'maximum count
  … (unscaled):',Hg.max(),'(',round(100*Hg.max()/lenscounterg,2),'% of
  … sample)'
180 #print 'maximum bin count (scaled)', scaledHgmax
181 print 'occurs in bin:'
182
```

```
183 maxHgzlbins=maxHgbins[0]
184 maxHgzsbins=maxHgbins[1]
185 maxHgb_siglsqdbins=maxHgbins[2]
186
187 #print 'maxcount zl bin no: ', maxHgzlbins
188 #print 'maxcount zs bin no: ' , maxHgzsbins
189 #print 'maxcount b_siglsqd bin no: ', maxHgb_siglsqdbins
190
191 print 'zl: ',  (edges[0][(maxHgzlbins)]),' - ',
    …(edges[0][(maxHgzlbins)])+zlbinsize
192 print 'zs: ',  (edges[1][(maxHgzsbins)]),' - ',
    …(edges[1][(maxHgzsbins)])+zsbinsize
193 print 'b_siglsqd : ',  (edges[2][(maxHgb_siglsqdbins)]),' - ',
    …(edges[2][(maxHgb_siglsqdbins)])+b_siglsqdbinsize
194 print
195
196 print 'COSMOLOGY DELTA (SC vs TC)'
197 print '==========================='
198
199 # Now we have H and Hg (ie. cosmology 1 and cosmology 2 histograms), the
    …delta is trivial:
200 deltaH = np.absolute(H-Hg)   # we need the modulus of the difference!
201
202 # Analyse the delta histogram;
203 maxdeltaHbins=np.where(deltaH==deltaH.max())    # identify coordinates
    …(zl, zs, b_siglsqd) of bin(s) with biggest difference in count
204 #print 'coordinates (zl, zs, b_siglsqd) of bin(s) with biggest
    …difference in count: ', maxdeltaHbins
205
206 #scaleddeltaHmax=scalefactor*deltaH.max()
207
208 print 'maximum difference count (unscaled):', deltaH.max()  # shows
    …highest difference in count = > greatest difference between cosmologies
209 #print 'Maximum bin count (scaled)', scaleddeltaHmax # scaled
210 print 'occurs in bin:'
211
212 maxdeltaHzlbins=maxdeltaHbins[0]   # returns position of zs bin(s)
    …corresponding to highest delta count
213 maxdeltaHzsbins=maxdeltaHbins[1]   # returns position of zl bin(s)
    …corresponding to highest delta count
214 maxdeltaHb_siglsqdbins=maxdeltaHbins[2]    # returns position of b
    …bin(s) corresponding to highest delta count
215
216 #print 'max delta zl bin no: ', maxdeltaHzlbins
217 #print 'max delta zs bin no: ' , maxdeltaHzsbins
218 #print 'max delta b_siglsqd bin no: ', maxdeltaHb_siglsqdbins
219
220 print 'zl: ',  (edges[0][(maxdeltaHzlbins)]),' - ',
    …(edges[0][(maxdeltaHzlbins)]) + zlbinsize # returns zs bin(s)
```

```
220… corresponding to highest difference
221 print 'zs: ',  (edges[1][(maxdeltaHzsbins)]),' - ',
… (edges[1][(maxdeltaHzsbins)]) + zsbinsize  # returns zl bin(s)
… corresponding to highest difference
222 print 'b_siglsqd : ',  (edges[2][(maxdeltaHb_siglsqdbins)]),' - ',
… (edges[2][(maxdeltaHb_siglsqdbins)]) + b_siglsqdbinsize    # returns b
… bin(s) corresponding to highest difference
223 print
224
225 print 'SIGNIFICANCE OF DEVIATION ("Si")'
226 print '================================='
227
228 nonzerobins=np.where(H+Hg>0)  # returns coords of all bins that are >0
… in one or other cosmology; use this to suppress bins that are always
… empty
229
230 # Calculate the Si value (Bityukov):
231 Hvol = H.sum()
232 Hgvol = Hg.sum()
233 K = Hvol/Hgvol  # normalization constant
234 # K=1  use this instead of previous line to cross-check Si difference
… below
235
236 netdeltaH = H[(nonzerobins)]-(K*Hg[(nonzerobins)])   # numerator of Si
237 NSnetdeltaH = H-(K*Hg)   #original non-suppressed code
238
239 HplusKsqdHg = H[(nonzerobins)] + ((K**2)*Hg[(nonzerobins)])
240 NSHplusKsqdHg = H + ((K**2)*Hg)  #original non-suppressed code
241 NSHplusKsqdHg[NSHplusKsqdHg == 0] = 1 # need to avoid division by zero
… (required in original non-suppressed code)
242
243 sigmaH=np.sqrt(HplusKsqdHg)  # denominator of Si; note definition of VAR
… for 2 Poisson distributions is just N1 + N2
244 NSsigmaH=np.sqrt(NSHplusKsqdHg)
245
246 SiH = netdeltaH/sigmaH  # see Bityukov for nomenclature
247 NSSiH = NSnetdeltaH/NSsigmaH
248
249 #the following requires the original non-suppressed code, as it uses the
… bin indices:
250 NSmaxSiHbins=np.where(abs(NSSiH)==abs(NSSiH).max()) # identify
… coordinates (zl, zs, b_siglsqd) of bin(s) with highest value of abs Si
251 NSmaxSiHzlbins=NSmaxSiHbins[0]   # returns zs bin(s) corresponding to
… highest Si
252 NSmaxSiHzsbins=NSmaxSiHbins[1]   # returns zl bin(s) corresponding to
… highest Si
253 NSmaxSiHb_siglsqdbins=NSmaxSiHbins[2]    # returns b_siglsqd bin(s)
… corresponding to highest Si
254
```

```
255 print 'maximum (absolute) Si occurs in SC and TC bin with respective
    … counts ',  (H[NSmaxSiHbins]),' and ', (Hg[NSmaxSiHbins])
256 #print '(cross-check with K-normalised difference: ',
    … (NSnetdeltaH[NSmaxSiHbins]),')'  # use with K=1 as cross-check.
257
258
259 print 'maximum (absolute) Si value:', abs(NSSiH).max()
260
261 #the following requires the original non-suppressed code, as it uses the
    … bin indices:
262 print 'occurs in bin:'
263 print 'zl: ',  (edges[0][(NSmaxSiHzlbins)]),' - ',
    … (edges[0][(NSmaxSiHzlbins)])+zlbinsize  # returns zl bin(s)
    … corresponding to highest Si
264 print 'zs: ',  (edges[1][(NSmaxSiHzsbins)]),' - ',
    … (edges[1][(NSmaxSiHzsbins)])+zsbinsize  # returns zs bin(s)
    … corresponding to highest Si
265 print 'b_siglsqd : ',  (edges[2][(NSmaxSiHb_siglsqdbins)]),' - ',
    … (edges[2][(NSmaxSiHb_siglsqdbins)])+b_siglsqdbinsize   # returns b
    … bin(s) corresponding to highest Si
266
267
268 print
269 numberofHbins = len(H.flatten())
270 numberofnonzerobins = len(H[(nonzerobins)].flatten())
271 print 'no of bins: ', numberofHbins
272 print 'no of non-zero bins: ', numberofnonzerobins
273 #print 'cross-check:',
    … (len(edges[0])-1)*(len(edges[1])-1)*(len(edges[2])-1)   # to cross-check
    … bin count
274 print
275
276 # calculate MEAN and RMS of Si
277 totalSi = SiH.sum()
278
279 meanSi = totalSi/numberofnonzerobins
280 print 'mean Si : ',round(meanSi,4)
281
282 Si_less_meanSi = SiH - meanSi
283 Si_less_meanSi_sqd = Si_less_meanSi**2
284 totalSi_less_meanSi_sqd = Si_less_meanSi_sqd.sum()
285
286 RMS = np.sqrt(totalSi_less_meanSi_sqd/numberofnonzerobins)
287 print 'RMS Si: ', round(RMS,4)
288
289 # calculate Chi_square values
290 print
291 '''
292 The following is test code for trying out scipy chi_square function;
```

```
useless as some counts < min reqd.
flatH = H.flatten()  # need to turn multi-dimensional array into a 1-D array
flatHg = Hg.flatten()

nonzeroflatH = flatH
nonzeroflatH[nonzeroflatH ==0] =1
#ChiSqd = stats.chisquare(flatH,f_exp=nonzeroflatHg)
ChiSqd = stats.chisquare(flatHg,f_exp=nonzeroflatH)
#print 'Scipy Chi square: ', ChiSqd
'''
# calculate Chi square manually
SiHsqd = SiH**2
totalSiHsqd=SiHsqd.sum()
meanSisqd = totalSiHsqd/numberofnonzerobins

print "CHI SQUARE"
print '============'
print 'Chi square: ', round(totalSiHsqd ,4)
print 'mean Chi square: ', round(meanSisqd,4)
#print '(cross-check RMS Si (Bityukov) from Chi square (Serjeant):', np.sqrt((totalSiHsqd/numberofnonzerobins)-(meanSi**2)),')'

print 'degrees of Freedom: ', numberofnonzerobins-1,'(non-zero);', numberofHbins-1,'(all bins)'
print
Ztest = (meanSisqd-1)/(np.sqrt(3.0/(numberofnonzerobins-1)))
print 'Z test : ', round(Ztest,4)
print '[with Gaussian of mean = 1 and standard deviation = ', round(np.sqrt(3.0/numberofnonzerobins),4),'(sqrt(3/',numberofnonzerobins-1,')]'
print
print 'P-value ("survival function"):', round(stats.norm.sf(abs(Ztest)),4)    # calculates p-value of ABSOLUTE Z
if Ztest < 0:
    print '(absolute value of Z test used, ie.', round(abs(Ztest),4),')'
print
print '(total sample count less total histogram count: ', lenscounterf+lenscounterg-H.sum()-Hg.sum(),')'
print
print 'END OF REPORT - run on: ', time.strftime("%d/%m/%Y")
```

E.3 Plotting Likelihood Curves

For the range of Ω_m values, it was relatively straightforward to plot the likelihoods (whether as p-values or Z-test values) when tested against the Planck best-fit flat ΛCDM cosmology. In fact, two versions of this plot were carried out. The first version was simply a raw plot of both the p-values and Z-test values that had been derived using code specifically written for this purpose[1]. However, the limited range of values and corresponding population sizes resulted in a somewhat 'jagged' appearance that at best was visually confusing; an example is shown in Figure E.5

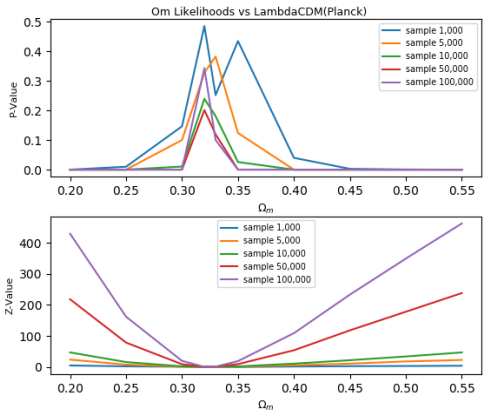

Figure E.5: Idealised Lenses & Likelihood Plots for Ω_m

In a second version, the Z-test values were fitted with a third-degree polynomial, and the corresponding p-values subsequently determined and plotted from that curve. However, fitting with a polynomial in this manner initially resulted in an anomaly, whereby across the full range of Ω_m the fitted curve did not allow the Z-test values to approach zero (which they do in the neighbourhood of the Concordance value). Upon closer inspection, it became apparent that a better fit could be obtained over a narrower range of values for Ω_m. Once the range was restricted, the fit produced a much smoother and informative set of likelihood plots, consistent with the expectation that the distributions would tend to centre around the Concordance value. An example of the source code is shown below; the 'smoothed' plots themselves are included in section 5.2.

[1] See Appendix E.2

```
"""
This program is designed to derive and plot smoothed Z-test values for cosmologies of varying
values of Omega_m when tested against the Concordance LambdaCDM (Planck best-fit) cosmology.

Smoothing of actual z-values is performed by fitting with a third order polynomial. For sample sizes
exceeding 10,000 the Om range has been reduced to enable a more accurate fit.

The corresponding p-values are then plotted (as Likelihoods)

Values are plotted based on different sample sizes for each cosmology.

Choose between plots of z-values (actual or smoothed) or p_values
(actual or derived) below.


CFW
9/7/2018

"""

import matplotlib.pyplot as plt
import numpy as np

plt.close()

# Data for plotting: default range is 0.20 < Omega_m < 0.55
Om_20_to_55 =
[0.20,0.25,0.30,0.31,0.32,0.321,0.322,0.323,0.325,0.326,0.327,0.328,0.329,0.33,0.34,0.35,0.40,0.45,0.50,0.55]

p_1000=[0,0.0098,0.1466,0.1491,0.4855,0.3513,0.1203,0.4659,0.2187,0.3045,0.4114,0.4915,0.4289,0.2522,0.227,0.4348,0.0399,0.0027,0.0005,0]

z_1000 =
[5,2.3,1.05,1.0401,0.0364,0.3818,1.174,0.0857,0.7765,0.5114,0.224,0.0212,0.1791,.6676,0.7487,0.1643,1.75,2.8,3.3,4.13]

p_5000=[0,0,0.1001,0.4905,0.3278,0.4863,0.4961,0.1728,0.2202,0.2167,0.3649,0.0594,0.3922,0.3824,0.3364,0.1239,0,0,0,0]

z_5000=[23.9,7.74,1.29,0.0238,0.45,0.0344,0.0098,0.943,0.7717,0.7833,0.3455,1.5599,0.2736,0.29,0.4224,1.16,5.65,10.9,18,22.8]

p_10000=[0,0,0.0104,0.4008,0.2395,0.2211,0.4977,0.4939,0.466,0.1696,0.4857,0.2015,0.1356,0.1811,0.3928,0.0257,0,0,0,0]
```

```
37
38 z_10000=
   …[47,16,2.3,0.2513,0.707,0.7685,0.0057,0.0153,0.0853,0.9557,0.0358,0.8362
   …,1.1005,0.9,0.272,1.95,10.9,22,34,47]
39
40 # The Omega_m range is reduced for larger sample sizes to enable more
   …accurate fitting.
41
42 Om_30_to_35 =
   …[0.30,0.31,0.32,0.321,0.322,0.323,0.325,0.326,0.327,0.328,0.329,0.33,0.
   …34,0.35]
43
44 p_50000=[0,0.0003,0.2015,0.3017,0.2660,0.2767,0.3947,0.4941,0.3492,0.
   …1072,0.3948,0.1194,0.0012,0]
45
46 z_50000=
   …[9.02,3.477,0.84,0.5194,0.625,0.5926,0.267,0.0149,0.3874,1.2417,0.2669,1
   ….1779,3.0313,9.37]
47
48 p_100000=[0,0,0.3435,0.2882,0.2106,0.2807,0.4212,0.1640,0.2864,0.0907,0.
   …0667,0.1001,0,0]
49
50 z_100000=
   …[19.8,5.682,0.403,0.5587,0.805,0.5807,0.1987,0.978,0.5638,1.3362,1.5006,
   …1.28,6.7852,19]
51
52 p_200000
   …=[0,0,0.0917,0.1208,0.3997,0.4691,0.4255,0.0227,0.328,0.0047,0.0308,0.
   …0003,0,0]
53
54 z_200000 =
   …[36.8,12.72,1.3307,1.1708,0.2542,0.0774,0.1877,2.0011,0.4455,2.5985,1.
   …8687,3.4049,14.3544,35.05]
55
56
57 # calculate polynomial fits – default Omega_m range
58 coeffs_1000 = np.polyfit(Om_20_to_55, z_1000, 3)   # this returns the
   …coefficients of the 3rd order polynomial best fit
59 newfit_1000 = np.poly1d(coeffs_1000)     # newfit(x) is a function that
   …passes x as the variable to the polynomial and returns the result y.
60
61 coeffs_5000 = np.polyfit(Om_20_to_55, z_5000, 3)
62 newfit_5000 = np.poly1d(coeffs_5000)
63
64 coeffs_10000 = np.polyfit(Om_20_to_55, z_10000, 3)
65 newfit_10000 = np.poly1d(coeffs_10000)
66
67 # calculate polynomial fits – reduced Omega_m range
68
```

```
coeffs_50000 = np.polyfit(Om_30_to_35, z_50000, 3)
newfit_50000 = np.poly1d(coeffs_50000)

coeffs_100000 = np.polyfit(Om_30_to_35, z_100000, 3)
newfit_100000 = np.poly1d(coeffs_100000)

coeffs_200000 = np.polyfit(Om_30_to_35, z_200000, 3)
newfit_200000 = np.poly1d(coeffs_200000)


# define new Om axis
x_new = np.linspace(0.2,0.50,1000)

# calculate new values for z based on polynnomial fit with corresponding
p values
z_1000_new = newfit_1000(x_new)
p_1000_new=np.exp(-0.5*(z_1000_new**2))

z_5000_new = newfit_5000(x_new)
p_5000_new=np.exp(-0.5*(z_5000_new**2))

z_10000_new = newfit_10000(x_new)
p_10000_new=np.exp(-0.5*(z_10000_new**2))

z_50000_new = newfit_50000(x_new)
p_50000_new=np.exp(-0.5*(z_50000_new**2))

z_100000_new = newfit_100000(x_new)
p_100000_new=np.exp(-0.5*(z_100000_new**2))

z_200000_new = newfit_200000(x_new)
p_200000_new=np.exp(-0.5*(z_200000_new**2))


# plot the curves/points
plt.plot(x_new, p_1000_new, label= 'sample = 1,000',color='red')
plt.plot(x_new, p_5000_new, label= 'sample = 5,000',color='blue')
plt.plot(x_new, p_10000_new, label= 'sample = 10,000',color='brown')
plt.plot(x_new, p_50000_new, label= 'sample = 50,000',color='green')
plt.plot(x_new, p_100000_new, label= 'sample = 100,000',color='orange')
plt.plot(x_new, p_200000_new, label= 'sample = 200,000',color='black')

plt.suptitle('Likelihoods vs LambdaCDM(Planck)',size=10)
plt.xlabel("$\Omega_m$",size = 9)
plt.ylabel("Likelihood",size=8)

'''
# use this routine to test the z fits (adjust for sample size):
plt.suptitle('Z-test vs LambdaCDM(Planck)',size=10)
```

```
117 plt.plot(Om,z_200000,'.')
118 plt.plot(Om,newfit_200000(Om))
119 plt.xlabel("$\Omega_m$",size = 9)
120 plt.ylabel("Z-test",size=8)
121 '''
122
123 '''
124 # use these routines for plotting either original z of
125 rp values ('jagged')
126
127 plt.plot(Om,p_1000,label='sample 1,000')
128 plt.plot(Om,p_5000, label='sample 5,000')
129 plt.plot(Om,p_10000, label='sample 10,000')
130 plt.plot(Om,p_50000, label='sample 50,000')
131 plt.plot(Om,p_100000, label='sample 100,000')
132 plt.plot(Om,p_200000, label='sample 200,000')
133
134
135 plt.plot(Om,z_1000,label='sample 1,000')
136 plt.plot(Om,z_5000, label='sample 5,000')
137 plt.plot(Om,z_10000, label='sample 10,000')
138 plt.plot(Om,z_50000, label='sample 50,000')
139 plt.plot(Om,z_100000, label='sample 100,000')
140 plt.plot(Om,z_200000, label='sample 200,000')
141 '''
142 #plt.xlim(0.2,0.8)
143 plt.legend()
144 plt.show()
145
```

E.4 Filtering on Redshift

The modifications required to impose a filter of zs - zl < 1 on the full sky of idealised lenses, as discussed in section 5.2.2, relate to 'Stage Two' and to 'Stage Three' of the model.

The corresponding amendments are shown in lines 15, 79, 105 & 112 of the *ModelAll.py* source code and in line 118 of the *MaleResults.py* source code, examples of which are shown below.

```
from __init__ import *
import cPickle
#import pyfits
import sys
import pylab as plt
import time
sigfloor=200

# ****** CAUTION : This ModelAll code contains a filter to exclude systems with zs - zl > 1 *******


L=LensSample(reset=False,sigfloor=sigfloor)  # cw removed cosmo = [0.3,0.7,0.7] argument as not required with Astropy

experiment="Euclid"
frac=15000.*1./42000.   # cw - amended to scale full sky to Euclid survey area.

a=20#SN threshold
b=3#Magnification threshold

c=1000
d=1000


#experiment="DES"
if len(sys.argv)>1:
    experiment=sys.argv[1]
    frac=float(sys.argv[2])
if len(sys.argv)>3:
    a=int(sys.argv[3])
    b=int(sys.argv[4])
    #c=int(sys.argv[5])
    #d=int(sys.argv[6])

firstod=1
nsources=1


surveys=[]

if experiment=="Euclid":
    surveys+=["Euclid"]
if experiment=="CFHT":
    surveys+=["CFHT"] #full coadd (Gaussianised)
if experiment=="CFHTa":
    surveys+=["CFHTa"] #dummy CFHT
if experiment=="DES":
```

```
    surveys+=["DESc"] #Optimal stacking of data
    surveys+=["DESb"] #Best Single epoch image
    surveys+=["DESa"] #full coadd (Gaussianised)
if experiment=="LSST":
    surveys+=["LSSTc"] #Optimal stacking of data
    surveys+=["LSSTb"] #Best Single epoch image
    surveys+=["LSSTa"] #full coadd (Gaussianised)
    #print "only doing LSSTc"


S={}
n={}
for survey in surveys:
    S[survey]=FastLensSim(survey,fractionofseeing=1)
    S[survey].bfac=float(2)
    S[survey].rfac=float(2)


t0=time.clock()

#for sourcepop in ["lsst","cosmos"]:
for sourcepop in ["lsst"]:
  chunk=0
  Si=0
  SSPL={}
  foundcount={}
  for survey in surveys:
      foundcount[survey]=0

  if sourcepop=="cosmos":
      nall=1100000
  elif sourcepop=="lsst":
      nall=11500000  # cw - amended to approximate full sky number of
…idealised lenses
  nall=int(nall*frac)
  print
  print 'CAUTION:'
  print 'Analysing ', frac,' of full sky idealised lenses'
  print 'and filtered on zs - zl < 1'
  print

  for i in range(nall):
    if i%10000==0:
        print "about to load"
        L.LoadLensPop(i,sourcepop)
        print i,nall

    if i!=0:
        if i%10000==0 or i==100 or i==300 or i==1000 or i==3000:
```

```
95              t1=time.clock()
96              ti=(t1-t0)/float(i)
97              tl=(nall-i)*ti
98              tl/=60#mins
99              hl=numpy.floor(tl/(60))
100             ml=tl-(hl*60)
101             print i,"%ih%im left"%(hl,ml)
102
103     lenspars=L.lens[i]
104
105     zltest=lenspars["zl"]            # cw for redshift filter
106     zstest=lenspars["zs"][1]
107
108     if lenspars["lens?"]==False:
109         del L.lens[i]
110         continue
111
112     if zstest-zltest >1:       # cw - additional filter on source-lens
…   redshifts
113         del L.lens[i]
114         continue
115
116     lenspars["rl"]["VIS"]=(lenspars["rl"]["r_SDSS"]+\
117
…   lenspars["rl"]["i_SDSS"]+lenspars["rl"]["z_SDSS"])/3
118     for mi in [lenspars["ml"],lenspars["ms"][1]]:
119         mi["VIS"]=(mi["r_SDSS"]+mi["i_SDSS"]+mi["z_SDSS"])/3
120
121
122
123
124     #if lenspars["zl"]>1 or lenspars["zl"]<0.2 or
…   lenspars["ml"]["i_SDSS"]<17 or lenspars["ml"]["i_SDSS"]>22:continue#
…   this is a CFHT compare quick n dirty test
125
126     lenspars["mag"]={}
127     lenspars["msrc"]={}
128     lenspars["mag"]={}
129     lenspars["msrc"]={}
130     lenspars["SN"]={}
131     lenspars["bestband"]={}
132     lenspars["pf"]={}
133     lenspars["resolved"]={}
134     lenspars["poptag"]={}
135     lenspars["seeing"]={}
136     lenspars["rfpf"]={}
137     lenspars["rfsn"]={}
138
139     lastsurvey="non"
```

```
140     for survey in surveys:
141
142
…   S[survey].setLensPars(lenspars["ml"],lenspars["rl"],lenspars["ql"],reset
…   =True)
143         for j in range(nsources):
144
…   S[survey].setSourcePars(lenspars["b"][j+1],lenspars["ms"][j+1],\
145
…   lenspars["xs"][j+1],lenspars["ys"][j+1],\
146
…   lenspars["qs"][j+1],lenspars["ps"][j+1],\
147                                     lenspars["rs"][j+1],sourcenumber=j+1
…   )
148
149         if survey[:3]+str(i)!=lastsurvey:
150             model=S[survey].makeLens(stochasticmode="MP")
151             SOdraw=numpy.array(S[survey].SOdraw)
152             if type(model)!=type(None):
153                 lastsurvey=survey[:3]+str(i)
154             if S[survey].seeingtest=="Fail":
155                 lenspars["pf"][survey]={}
156                 lenspars["rfpf"][survey]={}
157                 for src in S[survey].sourcenumbers:
158                     lenspars["pf"][survey][src]=False
159                     lenspars["rfpf"][survey][src]=False
160                 continue#try next survey
161         else:
162             S[survey].loadModel(model)
163             S[survey].stochasticObserving(mode="MP",SOdraw=SOdraw)
164             if S[survey].seeingtest=="Fail":
165                 lenspars["pf"][survey]={}
166                 for src in S[survey].sourcenumbers:
167                     lenspars["pf"][survey][src]=False
168                 continue#try next survey
169             S[survey].ObserveLens()
170
171
…   mag,msrc,SN,bestband,pf=S[survey].SourceMetaData(SNcutA=a,magcut=b,
…   SNcutB=[c,d])
172         lenspars["SN"][survey]={}
173         lenspars["bestband"][survey]={}
174         lenspars["pf"][survey]={}
175         lenspars["resolved"][survey]={}
176         lenspars["poptag"][survey]=i
177         lenspars["seeing"][survey]=S[survey].seeing
178         rfpf={}
179         rfsn={}
180         for src in S[survey].sourcenumbers:
```

```
181             rfpf[src]=False
182             rfsn[src]=[0]
183             lenspars["mag"][src]=mag[src]
184             lenspars["msrc"][src]=msrc[src]
185             lenspars["SN"][survey][src]=SN[src]
186             lenspars["bestband"][survey][src]=bestband[src]
187             lenspars["pf"][survey][src]=pf[src]
188             lenspars["resolved"][survey][src]=S[survey].resolved[src]
189         if survey!="Euclid":
190             if S[survey].seeingtest!="Fail":
191                 if survey not in ["CFHT","CFHTa"]:
192
…   S[survey].makeLens(noisy=True,stochasticmode="1P",SOdraw=SOdraw,
…   MakeModel=False)
193
…   rfpf,rfsn=S[survey].RingFinderSN(SNcutA=a,magcut=b,SNcutB=[c,d],mode="
…   donotcrossconvolve")
194                 else:
195
…   rfpf,rfsn=S[survey].RingFinderSN(SNcutA=a,magcut=b,SNcutB=[c,d],mode="
…   crossconvolve")
196         lenspars["rfpf"][survey]=rfpf
197         lenspars["rfsn"][survey]=rfsn
198
199         ###
200         #This is where you can add your own lens finder
201         #e.g.
202         #found=Myfinder(S[survey].image,S[survey].sigma,\
203         #                    S[survey].psf,S[survey].psfFFT)
204         #NB/ image,sigma, psf, psfFFT are dictionaries
205         #    The keywords are the filters, e.g. "g_SDSS", "VIS" etc
206
207         #then save any outputs you'll need to the lenspars dictionary:
208         #lenspars["my_finder_result"]=found
209
210         ###
211
212         #If you want to save the images (it may well be a lot of data!):
213         #import pyfits #(or the astropy equivalent)
214
215         #folder="where_to_save_fits_images"
216         #folder="%s/%i"%(folder,i)
217         #for band in S[survey].bands:
218             #img=S[survey].image[band]
219             #sig=S[survey].sigma[band]
220             #psf=S[survey].psf[band]
221             #resid=S[survey].fakeResidual[0][band]#The lens subtracted
222
223         #resid contains the lensed source, with the lens subtracted
```

```
224         #assuming the subtraction is poisson noise limited (i.e. ideal)
225
226
…  #pyfits.PrimaryHDU(img).writeto("%s/image_%s.fits"%(folder,band),\
227                 #                                clobber=True)
228
…  #pyfits.PrimaryHDU(sig).writeto("%s/sigma_%s.fits"%(folder,band),\
229                 #                                clobber=True)
230
…  #pyfits.PrimaryHDU(psf).writeto("%s/psf_%s.fits"%(folder,band),\
231                 #                                clobber=True)
232
…  #pyfits.PrimaryHDU(resid).writeto("%s/galsub_%s.fits"%(folder,band),
…  clobber=True)
233
234         ###
235
236         L.lens[i]=None #delete used data for memory saving
237
238     accept=False
239     for survey in surveys:
240         if lenspars["pf"][survey][1]:
241             accept=True
242
243     if accept:
244         #S[survey].display(band="VIS",bands=["VIS","VIS","VIS"])
245         #if Si>100:exit()
246         Si+=1
247         SSPL[Si]=lenspars.copy()
248         if (Si+1)%1000==0:
249
…  f=open("LensStats/%s_%s_Lens_stats_%i.pkl"%(experiment,sourcepop,chunk),
…  "wb")
250             cPickle.dump([frac,SSPL],f,2)
251             f.close()
252             SSPL={} # reset SSPL or memory fills up
253             chunk+=1
254
255     del L.lens[i]
256
257
…  f=open("LensStats/%s_%s_Lens_stats_%i.pkl"%(experiment,sourcepop,chunk),
…  "wb")
258   cPickle.dump([frac,SSPL],f,2)
259   f.close()
260   print Si
261
262 bl=[]
263 for j in SSPL.keys():
```

```
264     try:
265         if SSPL[j]["rfpf"][survey][1]:
266             bl.append(SSPL[j]["b"][1])
267     except KeyError:pass
268
```

```python
from __init__ import *
import cPickle
#import pyfits
import sys,os
import pylab as plt
import glob

params = {
   'axes.labelsize': 14,
   'text.fontsize': 14,
   'legend.fontsize': 10,
   'xtick.labelsize': 10,
   'ytick.labelsize': 10,
   'text.usetex': False,
    'figure.figsize': [6, 4]
   }
plt.rcParams.update(params)

sourcepops=["lsst"]

experiment="Euclid"
#experiment="CFHT"
#experiment="LSST"
#experiment="DES"

if len(sys.argv)>1:
    experiment=sys.argv[1]

surveystoread=[]
if experiment=="Euclid":
    surveystoread+=["Euclid"]
elif experiment=="CFHT":
    surveystoread+=["CFHT"]
elif experiment=="CFHTa":
    surveystoread+=["CFHTa"]
elif experiment=="DES":
    surveystoread+=["DESc"]
    surveystoread+=["DESb"]
    surveystoread+=["DESa"]
elif experiment=="LSST":
    surveystoread+=["LSSTc"]
    surveystoread+=["LSSTb"]
    surveystoread+=["LSSTa"]
else:
    surveystoread=[str(experiment)]
    experiment=experiment[:-1]


for survey in surveystoread:
```

```
50  for sourcepop in sourcepops:
51        # if survey[-2]=="a":       cw - [-2] is wrong element position
52    #    surveyname=survey[:-1]+"_full_coadd"
53    #elif survey[-2]=="b":
54     #   surveyname=survey[:-1]+"_best_epoch"
55    #elif survey[-2]=="c":
56     #   surveyname=survey[:-1]+"_optimal_coadd"
57    #else:
58    surveyname=survey  # cw - removed indent as no 'if' clause now
59    filename="%s_%s_lists.pkl"%(survey,sourcepop)
60    lensparsfile="lenses_%s.txt"%survey
61    f=open(lensparsfile,"w")
62    print
63    #os.system("rm %s"%filename) #this line resets the read-in
64    bl={}
65    zs={}
66    zl={}
67    sigl={}
68    ql={}
69    rs={}
70    ms={}
71    mag={}
72    weights={}
73    for key in ["resolved","rfpf"]:
74        bl[key]=[]
75        zs[key]=[]
76        rs[key]=[]
77        ms[key]=[]
78        zl[key]=[]
79        sigl[key]=[]
80        ql[key]=[]
81        mag[key]=[]
82        rs[key]=[]
83        weights[key]=[]
84
85    if experiment=="CFHT":
86      frac=42000.*1./150.
87      bands=["g_SDSS","r_SDSS","i_SDSS"]
88
89    if experiment=="CFHTa":
90      frac=42000.*1./150.
91      bands=["g_SDSS","r_SDSS","i_SDSS"]
92
93    elif experiment=="Euclid":
94      #frac=42000.*1./15000.
95      frac=1.0
96      bands=["VIS"]
97
98    elif experiment=="DES":
```

```
99        frac=42000.*1./5000.
100       bands=["g_SDSS","r_SDSS","i_SDSS"]
101
102     elif experiment=="LSST":
103       frac=42000.*1./20000.
104       bands=["g_SDSS","r_SDSS","i_SDSS"]
105
106
107
… filelist=glob.glob("LensStats/%s_%s_Lens_stats_*.pkl"%(experiment,
… sourcepop))
108
109     chunki=0
110     ilist=[]
111     print survey
112     for chunk in filelist:
113         print chunki
114         chunki+=1
115         f2=open(chunk,"rb")
116         fracsky,sspl=cPickle.load(f2)
117         #fract=frac*fracsky
118         fract = 1.0  # cw - amended as sample already scaled.
119         f2.close()
120         I=0
121         for i in sspl.keys():
122             if i in ilist:
123                 continue
124             else:
125                 try:
126                     sspl[i]["seeing"][survey]
127                 except KeyError:
128                     continue
129                 f.write("%.2f "%sspl[i]["zl"])
130                 f.write("%.2f "%sspl[i]["zs"][1])
131                 f.write("%.2f "%sspl[i]["b"][1])
132                 f.write("%.2f "%sspl[i]["sigl"])
133                 f.write("%.2f "%sspl[i]["ql"])
134                 f.write("%.2f "%sspl[i]["rl"]["g_SDSS"])
135                 for band in bands:
136                     f.write("%.2f "%sspl[i]["ml"][band])
137               # f.write("%.2f "%sspl[i]["rl"]["g_SDSS"])   CW -
… commented out as duplicate element
138                 f.write("%.2f "%sspl[i]["xs"][1])
139                 f.write("%.2f "%sspl[i]["ys"][1])
140                 f.write("%.2f "%sspl[i]["qs"][1])
141                 f.write("%.2f "%sspl[i]["ps"][1])
142                 f.write("%.2f "%sspl[i]["rs"][1])
143                 for band in bands:                                  # CW -
… this line and line below added to include ms in txt file
```

```
144                     f.write("%.2f "%sspl[i]["ms"][1][band])
145                 f.write("%.2f "%sspl[i]["mag"][1])
146                 for band in bands:
147                     f.write("%.2f "%sspl[i]["seeing"][survey][band])
148                     f.write("%.2f "%sspl[i]["SN"][survey][1][band][0])
149                 if survey!="Euclid":
150                     f.write("%.2f "%sspl[i]["rfsn"][survey][1][0])
151                 f.write("\n")
152
153
154                 ilist.append(str(i))
155                 if sspl[i]["pf"][survey][1]==False:continue
156
157                 try:
158                     bb=sspl[i]["bestband"][survey][1]
159                     #print sspl[i]["seeing"][survey][bb]
160                     #print sspl[i]["mag"][1]*sspl[i]["rs"][1],
161                     try:
162                         (sspl[i]["b"][1]**2-sspl[i]["rs"][1]**2)**0.5
163                     except FloatingPointError: print 0
164                 except KeyError:
165                   pass
166                 try:
167                   if
…  sspl[i]["resolved"][survey][1][sspl[i]["bestband"][survey][1]]:
168                     bb=sspl[i]["bestband"][survey][1]
169                     if sspl[i]["mag"][1]<3:continue
170                     if sspl[i]["SN"][survey][1][bb][0]<20:continue
171
172                     bl["resolved"].append(sspl[i]["b"][1])
173                     weights["resolved"].append(1./fract)
174                     zs["resolved"].append(sspl[i]["zs"][1])
175                     rs["resolved"].append(sspl[i]["rs"][1])
176                     zl["resolved"].append(sspl[i]["zl"])
177                     sigl["resolved"].append(sspl[i]["sigl"])
178                     ql["resolved"].append(sspl[i]["ql"])
179                     mag["resolved"].append(sspl[i]["mag"][1])
180                     ms["resolved"].append(sspl[i]["ms"][1]["g_SDSS"])
181
182                     if sspl[i]["rfpf"][survey][1]:
183                         if sspl[i]["rfsn"][survey][1][0]<20:continue
184                         if
…  sspl[i]["resolved"][survey][1]["RF"]==False:continue
185
186                         if experiment=="CFHT" or experiment=="CFHTa":
187                             if sspl[i]["zl"]>1:continue
188                             if sspl[i]["zl"]<0.2:continue
189                             if sspl[i]["ml"]["i_SDSS"]<17:continue
190                             if sspl[i]["ml"]["i_SDSS"]>22:continue
```

```
191
192                         bl["rfpf"].append(sspl[i]["b"][1])
193                         weights["rfpf"].append(1./fract)
194                         zs["rfpf"].append(sspl[i]["zs"][1])
195                         rs["rfpf"].append(sspl[i]["rs"][1])
196                         zl["rfpf"].append(sspl[i]["zl"])
197                         sigl["rfpf"].append(sspl[i]["sigl"])
198                         ql["rfpf"].append(sspl[i]["ql"])
199                         mag["rfpf"].append(sspl[i]["mag"][1])
200                         ms["rfpf"].append(sspl[i]["ms"][1]["g_SDSS"])
201
202
203                 except KeyError:
204                   pass
205     f.close()
206
207 # if survey[-2]=="a":     cw - [-2] is wrong character position
208     #    surveyname=survey[:-1]+" (full coadd)"
209     #elif survey[-2]=="b":
210     #    surveyname=survey[:-1]+" (best single epoch imaging)"
211     #elif survey[-2]=="c":
212     #    surveyname=survey[:-1]+" (optimal coadd)"
213     #else:
214     surveyname=survey #cw - removed indent as not part of if clause now
215
216
217     print survey, "will find",
218     print numpy.sum(numpy.array(weights["resolved"]).ravel()),
219     print "lenses assuming poisson limited galaxy subtraction in all
…  bands, or",
220     print numpy.sum(numpy.array(weights["rfpf"]).ravel()),
221     print "lenses in the g-i difference images"
222
223     f=open(filename,"wb")
224     cPickle.dump([weights,bl,zs,rs,ms,zl,sigl,ql,mag],f,2)
225     f.close()
226
227
228
229 bson=numpy.array([2.66,1.24,1.27,2.39,1.41,1.27,1.00,1.3,1.0,1.19,1.22,1
…  .36,1.76,1.19,1.29,1.56,1.04,0.85,1.10,1.23,1.16,0.93,1.03,1.4,0.74,1.21
…  ,1.14,1.74,2.03,1.23,2.55,1.05,1.51,4.36,0.94,0.93,3.11,1.79,0.96,1.40,1
…  .3,0.81,1.95,1.66,1.55,1.07,1.06,1.38,0.52,2.16,1.40,1.44])
230 plt.hist(bson,bins=numpy.linspace(0,3,16),weights=bson*0+220./len(bson),
…  fc="grey",alpha=0.6)
231 a,b=numpy.histogram(bl["rfpf"],bins=numpy.linspace(0,3,31),weights=
…  weights["rfpf"])
232 a*=2#double for finer bins
233 plt.plot(b[:-1]+(b[1]-b[0])/2.,a,c="k",lw=3,ls="dashed")
```

```
234 plt.xlabel(r"$\Theta_\mathrm{E}$ (arcsec)")
235 plt.ylabel(r"Lenses per $\Theta_\mathrm{E}$ bin")
236 plt.tight_layout()
237 plt.show()
238
```

Appendix F

Submillimetre Galaxies

F.1 Creating a Mock Catalogue

The script written to import the data from Cai et al. (2013) and Ikarashi et al. (2015) and to create a mock catalogue of submillimetre galaxies (SMGs) is shown below. The routine creates a catalogue of 100,000 simulated galaxies.

```
'''
Importing Cai data:-

This program has been designed to import the submm galaxy data provided by Cai from his "save" files.
It saves the data to a txt file called "textfileCai".

Based on the data read in above, the code creates a PDF as dNdlogS (ie. dN/dlogSdz x dz) vs logS;
ie. for each logS, add the [dN/dlogS evaluated at each z] over all the z.

Multiplying the cumulative dN/dlogS's by 0.06 (= dlogS) gives the dN for each logS and a cumulative total of
the dN's gives the CDF; this is un-normalised and needs to be normalised to produce a true CDF.

For each logS, sums the d3NdlogSdzdo across the redshifts and multiplies by dlogS and dz to get dN.

Simulation of submm data:-

(i) The code outputs the logS and redshift values based on the respective dN distributions.

(ii) Angular sizes are derived from probability distributions in Fig 6 of Ikarashi ('Compact Starbursts ...').

(iii) The resultant MOCK CATALOGUE is output to a file called "submmdataCai.txt".

Note: SAMPLE SIZE needs to be amended as required - default 100,000.


CFW
18/3/2019
'''

from scipy.io.idl import readsav     # need this to import Cai data (which is in IDL format)
import scipy.interpolate             # need this for interpolation of logS CDF
import numpy as np
import matplotlib.pyplot as plt

# This import registers the 3D projection, but is otherwise unused.
from mpl_toolkits.mplot3d import Axes3D

# read in saved data as file called 'all'
```

```
39 all=readsav('/Users/charles/Desktop/submm
   … galaxies/cai_saves/d3NdlogSdzdo_sph_tot.save')   # use 'copy path'
   … facility in Canopy
40
41 # create file called CAT which will be the catalogue structure
42 cat=all.d3NdlogSdzdo_0
43
44 # create files called WAVE_OBS and FILTER containing observed
   … wavelengths and filters respectively
45 wave_obs=cat.wave_obs   # this is a size 68 array
46 filter=cat.FID          # also a size 68 array
47
48 '''
49  use the wave_obs and filter files to search for index corresponding to
   … required band and wavelength
50  eg. " result = np.where(filter == 'SPIRE_500') " returns indexes 38,
   … 39, 40; then eg. wave_obs[39] = 500.0
51  use those indexes to run FLUXFUNC on the corresponding indexes in
   … "cat=all.d3NdlogSdzdo_0";
52  thus:
53 '''
54
55 d3NdlogSdzdo=[]   # initialise arrays
56 zp=[]
57 dz=[]
58 logS_nu = []
59
60 d3NdlogSdzdo=cat[39].FLUXFUNC  # choose eg. index 39 (500 micron); this
   … returns 201x181 array (units of galaxy_no/dex/dz/sr).
61
62 # d3NdlogSdzdo is a 2-d array of redshift and flux density, with
   … redshift and flux density from:
63 zp=cat[39].zp    # redshift; this is a size 181 array
64 dz=cat[39].dz    # redshift bins; this is a size 181 array
65 logS_nu=cat[39].LOGSNU  # flux density; this is a size 201 array (units
   … of log mJy)
66
67 '''
68 grid= np.meshgrid(zp,logS_nu)  # use this to create a grid for a plot;
   … returns a list of tuples, the elements of which are arrays.
69 '''
70
71 textfile="textfileCai.txt"   # set up file for storing initial Cai data
72 CDFfile="CDFfileCai.txt"    # set up file to be used for PDF and CDF
73
74 f=open(textfile,"w")
75
76 f.write('#'  '   d3N/dlogSdzdo        logS_nu       redshift')   #
   … creates a heading.
```

```
77  f.write("\n")                                                  # creates a
    …  new line
78
79  for i in range(0,len(logS_nu)):
80      for j in range(0,len(zp)):
81     #     print format(int(d3NdlogSdzdo[i][j]),','),'          ',
    …  logS_nu[i],'             ',    zp[j]
82     # Return/write data in the following order: (i) d3NdlogSdzdo (ii)
    …  logS_nu (iii) redshift, to textfileCai.txt
83          f.write("%.2f "%d3NdlogSdzdo[i][j])
84          f.write("%.2f "%logS_nu[i])
85          f.write("%.2f "%zp[j])
86          f.write("\n")
87
88  f.close()
89  print
90  print 'export of initial Cai data d3N/dlogSdzdo, logS_nu, redshift, to
    …  textfileCai.txt completed'
91
92  print
93
94  g=open(CDFfile,"w")
95  g.flush()
96  g.write('#'  'cumN    cumdN/dlogS   dN/dlogS  logS ')       # writes a
    …  heading to CDFfileCai.txt.
97  g.write("\n")
98
99  dNdlogSallz=[]  # initialise array
100
101 for s in range (0,len(logS_nu)):
102     dNsum=0
103     for z in range(0,len(zp)):
104         dNdlogS=d3NdlogSdzdo[s][z]*0.05    # dz = 0.05
105         dNsum+=dNdlogS   # adds up the dN/dlogS for logS_nu[s] over all
    …  redshifts
106
107     dNdlogSallz=np.append(dNdlogSallz,dNsum)    # creates an array of
    …  'dN/dlogS summed over z' corresponding to the array of logS
108     cumsumdNdlogS=dNdlogSallz.cumsum()         # creates running total
    …  of dN/dlogS
109     cumsumN=cumsumdNdlogS*0.06   # multiply by dlogS = 0.06 to give
    …  un-normalised CDF at each logS
110 #    print 'cumN:  ',cumsumN[s],'   cumdNdlogS (all z):
    …  ',cumsumdNdlogS[s],'  dNdlogS (all z): ',round(dNsum),'  logS: ',
    …  logS_nu[s]
111     g.write("%.2f "%cumsumN[s])    # exports cumsumN (un-normalised CDF)
    …  to CDFfileCai.txt
112     g.write("%.2f "%cumsumdNdlogS[s])  # exports cumsum of dNd/logS to
    …  CDFfileCai.txt
```

```
113     g.write("%.2f "%dNsum)  # exports dN/dlogS to CDFfileCai.txt
114     g.write("%.2f "%logS_nu[s])  # exports logS to CDFfileCai.txt
115     g.write("\n")
116 
117 g.close()
118 
119 print 'export of cumN, cumdN/dlogS, dN/dlogS, logS_nu, to CDFfileCai
    … (Stage Three - un-normalised CDF) completed'
120 
121 
122 # Now we create and write the CDF to a file called normedCDFCai.txt
123 
124 h = open("normedCDFCai.txt","w")
125 
126 h.write('#'  'CDF        logS_nu')   # creates a heading.
127 h.write("\n")                          # creates a new line
128 
129 for x in range(0,len(logS_nu)):
130     normedCDF = cumsumN/max(cumsumN)
131     minCDF=normedCDF.min()
132     h.write("%.4f "%normedCDF[x])
133     h.write("%.2f "%logS_nu[x])
134     h.write("\n")
135 h.close()
136 
137 print
138 print 'export of CDF, logS_nu, to normedCDFCai (Stage Four) completed'
139 print
140 
141 logS_result=[]   # initialise array
142 logS_resultarray=[]
143 
144 interpolated_logS_result=[]
145 interpolated_logS_resultarray=[]
146 
147 submm=open("submmdataCai.txt","w")       # initialise data txt file
148 submm.flush()
149 submm.write('# Galaxy  logS (mJy)  interpolated logS (mJy) redshift
    … rs_FWHM (arcsec)')       # writes a heading to CDFfileCai.txt.
150 submm.write("\n")
151 submm.close()
152 
153 for galcount in range (100000):      # SET FOR 100,000 GALAXIES  !!!
154     galnumber=galcount+1
155 #    print
156     if galnumber%10000==0:
157         print 'creating catalogue - galaxy number so far: ',galcount+1
158     prob=(1-minCDF)*np.random.random_sample()+minCDF   # Need this to
    … ensure random CDF prob is not below minimum of CDF probs.
```

```
159     probCDFindices=np.where(normedCDF<=prob)[0]
160     probCDFindex=probCDFindices.max()
161 
162     logS_result=logS_nu[probCDFindex]
163     logS_resultarray=np.append(logS_resultarray,logS_result)
164 
165     logS_interpol = scipy.interpolate.interp1d(normedCDF,logS_nu)
166     interpolated_logS_result=logS_interpol(prob)
167 
…   interpolated_logS_resultarray=np.append(interpolated_logS_resultarray,
…   interpolated_logS_result)
168 
169     logSindex=[]
170     logSindex=np.where(logS_nu==logS_result)[0]
171     dNsumz=0
172     dNperzarray=[]
173     for t in range (0,len(zp)):
174         dNperz=d3NdlogSdzdo[logSindex[0]][t]*0.05*0.06    # dz = 0.05,
…   dlogS=0.06
175         dNperzarray=np.append(dNperzarray,dNperz)   # an array of dN's
…   across all the redshifts for that logS_result
176         dNsumz+=dNperz   # adds up the dN for specified logS_nu over all
…   redshifts
177     PDFonz=dNperzarray/dNsumz   # creates PDF (prob. distribution of
…   redshifts for that logS_result)
178     CDFonz=PDFonz.cumsum()     # creates CDF
179     minCDFz=CDFonz.min()
180     probz=(1-minCDFz)*np.random.random_sample()+minCDFz
181     probCDFzindices=np.where(CDFonz<=probz)[0]
182     probCDFzindex=probCDFzindices.max()
183     z_result=zp[probCDFzindex]
184 
185     randindex=np.random.random()   # cw - create a random index number
…   for rs routine; major correction (no 2) introduced on 11/3/2019 !!!
186 
187     if z_result>=3:    # introduced equalities here and below on 11/3/19
188         if randindex<=(9.0/13.0):
189             rs=(0.3-0.05)*np.random.random_sample()+0.05   # cw -
…   adjusted to allow for min rs = 0.05
190         else:
191             rs=(0.5-0.3)*np.random.random_sample()+0.3
192     else:
193         if randindex>(9.0/12.0):     # cw - major correction (no 3!) to
…   logic in this routine on 12/3/2019
194             rs=(0.9-0.7)*np.random.random_sample()+0.7
195         if randindex<=(9.0/12.0) and randindex>(5.0/12.0):      # cw -
…   major corrrection (no 1!) made to reverse original order of these three
…   conditions !!!
196             rs=(0.7-0.5)*np.random.random_sample()+0.5
```

```
197         if randindex<=(5.0/12.0) and randindex>(2.0/12.0):
198             rs=(0.5-0.3)*np.random.random_sample()+0.3
199         if randindex<=(2.0/12.0):
200             rs=(0.3-0.05)*np.random.random_sample()+0.05   # cw -
… adjusted to allow for min rs = 0.05
201
202  #   print 'LogS - Probability: ', prob, '    CDF percentile:
… ',normedCDF[probCDFindex], '  Result (discretized) for
… logS_nu:',logS_result
203  #   print '   *** Interpolated  Result for logS_nu:  ',
… interpolated_logS_result,' ***'
204     # print 'INTERPOLATION TEST:
… ',logS_result+((prob)-normedCDF[probCDFindex])*0.06/(normedCDF[
… probCDFindex+1]-normedCDF[probCDFindex])
205  #   print 'Redshift - Probability:', probz, ', CDF percentile:
… ',CDFonz[probCDFzindex], ' *** Result for redshift:',z_result,' ***'
206
207     with open("submmdataCai.txt","a") as submm:
208         submm.write("%.0f "%galnumber)
209         submm.write("%.2f "%logS_result)    # Export simulated data to
… submmdataCai.txt file
210         submm.write("%.4f "%interpolated_logS_result)
211         submm.write("%.2f "%z_result)
212         submm.write("%.4f "%rs)
213         submm.write("\n")
214
215
216 print
217 print 'Simulated data for logS_nu, redshift, and angular sizes
… completed'
218
219
```

F.2 Loading the Mock Catalogue

The script written to load the SMG mock catalogue data into the model is shown below; the routine is referred to within the code as *loadsubmm* and is executed in place of the default routine *loadlsst*.

```
    def loadsubmm(self):                    # cw - introduce submm routine
        self.population="submm"
        data = 'submmdataCai.txt'

        g=open(data,"r")

        paramlist=[]    #initialises what will become a list of lists; that is, a list of data rows (each row being a list of three data)
        zp=[]           #initialise data
        logS_nu=[]
        rsdata=[]
        self.zc=[]
        self.logS=[]
        self.rc =[]

        for line in g:
            if line.startswith('#'):
                continue
            elements=line.split()    # reads in the data points for each row as a list (eg. three data points per row)
            paramlist.append(elements)    # builds up a list of 'lists' - ie. a list containing all of the data rows

        g.close()

        for x in range (len(paramlist)-1):    # loops through each 'list' in the list of 'lists' and builds up a list of individual parameters
            zp.append(paramlist[x][3])
            logS_nu.append(paramlist[x][2])
            rsdata.append(paramlist[x][4])
        self.zc=numpy.array(zp,float)   # cw - use this for import of zc rather than routine below
        self.logS=numpy.array(logS_nu,float)
        self.rc=0.5*numpy.array(rsdata,float)   # cw - *** angular size imported as FWHM so need to halve as half-light radius required***

        #self.zc=data[:,2]   # cw  - import redshift from PKL file here; but for txt file, use routine above.
        self.m={}

        self.m["g_SDSS"]=self.logS   # cw - amended to import log FLUX instead of m's
        self.m["r_SDSS"]= self.logS
        self.m["i_SDSS"]=self.logS
        self.m["z_SDSS"]=self.logS
        self.m["F814W_ACS"]=self.logS # we'll make do with F814==i
        self.m["Y_UKIRT"]=self.logS   #there is no Y band data atm
        self.mstar=[]  # cw - these data not needed
```

```
        self.mhalo=[]   #   cw -   --- "  ----
        self.m["VIS"]=self.logS
        self.rs=self.rc        #  cw -  import angular size here as rs


# CAN IGNORE FOLLOWING ROUTINE AS ANGULAR SIZES FOR SOURCES WILL BE
IMPORTED

#    def RofMz(self,M,z,scatter=True,band=None):#band independent so far
    #{mosleh et al}, {Huang, Ferguson et al.}, Newton SLACS XI.
 #       r_phys=((M/-19.5)**-0.22)*((1.+z)/5.)**(-1.2)
        # is the same as
 #       R=-(M+18.)/4.
 #       r_phys=(10**R)*((1.+z)/1.6)**(-1.2)
```

Bibliography

Abbott, T., Abdalla, F. B., Aleksić, J., Allam, S., Amara, A., Bacon, D., Balbinot, E., Banerji, M., Bechtol, K., Benoit-Lévy, A. et al. (2016), 'The dark energy survey: more than dark energy–an overview', *Monthly Notices of the Royal Astronomical Society* **460**(2), 1270–1299.

Abdo, A. (n.d.), 'Gravitational lensing'.

Abell, P. A., Burke, D. L., Hamuy, M., Nordby, M., Axelrod, T. S., Monet, D., Vrsnak, B., Thorman, P., Ballantyne, D., Simon, J. D. et al. (2009), Lsst science book, version 2.0, Technical report.

Alcock, C., Allsman, R., Alves, D., Axelrod, T., Becker, A., Bennett, D., Cook, K., Freeman, K., Griest, K., Guern, J. et al. (1997), 'The macho project large magellanic cloud microlensing results from the first two years and the nature of the galactic dark halo', *The Astrophysical Journal* **486**(2), 697.

Amvrosiadis, A., Valiante, E., Gonzalez-Nuevo, J., Maddox, S., Negrello, M., Eales, S., Dunne, L., Wang, L., van Kampen, E., De Zotti, G. et al. (2018), 'Herschel-atlas: the spatial clustering of low-and high-redshift submillimetre galaxies', *Monthly Notices of the Royal Astronomical Society* **483**(4), 4649–4664.

Astropy Collaboration, Robitaille, T. P., Tollerud, E. J., Greenfield, P., Droettboom, M., Bray, E., Aldcroft, T., Davis, M., Ginsburg, A., Price-Whelan, A. M., Kerzendorf, W. E., Conley, A., Crighton, N., Barbary, K., Muna, D., Ferguson, H., Grollier, F., Parikh, M. M., Nair, P. H., Unther, H. M., Deil, C., Woillez, J., Conseil, S., Kramer, R., Turner, J. E. H., Singer, L., Fox, R., Weaver, B. A., Zabalza, V., Edwards, Z. I., Azalee Bostroem, K., Burke, D. J., Casey,

A. R., Crawford, S. M., Dencheva, N., Ely, J., Jenness, T., Labrie, K., Lim, P. L., Pierfederici, F., Pontzen, A., Ptak, A., Refsdal, B., Servillat, M. & Streicher, O. (2013), 'Astropy: A community Python package for astronomy', *aap* **558**, A33.

Auger, M., Treu, T., Bolton, A., Gavazzi, R., Koopmans, L., Marshall, P., Bundy, K. & Moustakas, L. (2009), 'The sloan lens acs survey. ix. colors, lensing, and stellar masses of early-type galaxies', *The Astrophysical Journal* **705**(2), 1099.

Bityukov, S., Krasnikov, N., Nikitenko, A. & Smirnova, V. (2013*a*), 'A method for statistical comparison of histograms', *arXiv preprint arXiv:1302.2651* .

Bityukov, S., Krasnikov, N., Nikitenko, A. & Smirnova, V. (2013*b*), 'On the distinguishability of histograms', *The European Physical Journal Plus* **128**(11), 143.

Blain, A. W., Smail, I., Ivison, R., Kneib, J.-P. & Frayer, D. T. (2002), 'Submillimeter galaxies', *Physics Reports* **369**(2), 111–176.

Bolton, A. S., Burles, S., Koopmans, L. V., Treu, T. & Moustakas, L. A. (2006), 'The sloan lens acs survey. i. a large spectroscopically selected sample of massive early-type lens galaxies', *The Astrophysical Journal* **638**(2), 703.

Bousso, R. (2012), 'The cosmological constant problem, dark energy, and the landscape of string theory', *arXiv preprint arXiv:1203.0307* .

Brault, F. & Gavazzi, R. (2015), 'Extensive light profile fitting of galaxy-scale strong lenses-towards an automated lens detection method', *Astronomy & Astrophysics* **577**, A85.

Bussmann, R., Pérez-Fournon, I., Amber, S., Calanog, J., Gurwell, M., Dannerbauer, H., De Bernardis, F., Fu, H., Harris, A., Krips, M. et al. (2013), 'Gravitational lens models based on submillimeter array* imaging of herschel**-selected strongly lensed sub-millimeter galaxies at z¿ 1.5', *The Astrophysical Journal* **779**(1), 25.

Cai, Z.-Y., Lapi, A., Xia, J.-Q., De Zotti, G., Negrello, M., Gruppioni, C., Rigby, E., Castex, G., Delabrouille, J. & Danese, L. (2013), 'A hybrid model for the evolution of galaxies and active galactic nuclei in the infrared', *The Astrophysical Journal* **768**(1), 21.

Choi, Y.-Y., Park, C. & Vogeley, M. S. (2007), 'Internal and collective properties of galaxies in the sloan digital sky survey', *The Astrophysical Journal* **658**(2), 884.

Collett, T. E. (2015), 'The population of galaxy galaxy strong lenses in forthcoming optical imaging surveys', *The Astrophysical Journal* **811**(1), 20.
URL: *http://stacks.iop.org/0004-637X/811/i=1/a=20*

Connolly, A., Peterson, J., Jernigan, J. G., Abel, R., Bankert, J., Chang, C., Claver, C. F., Gibson, R., Gilmore, D. K., Grace, E. et al. (2010), Simulating the lsst system, *in* 'Modeling, Systems Engineering, and Project Management for Astronomy IV', Vol. 7738, International Society for Optics and Photonics, p. 77381O.

Dyson, F. (1920), 'Fw dyson, as eddington, and c. davidson, phil. trans. r. soc. a 220, 291 (1920).', *Phil. Trans. R. Soc. A* **220**, 291.

Eales, S. A. (2015), 'Practical cosmology with lenses', *Monthly Notices of the Royal Astronomical Society* **446**(3), 3224–3234.
URL: *http://dx.doi.org/10.1093/mnras/stu2214*

Eddington, A. S. (1919), 'The total eclipse of 1919 may 29 and the influence of gravitation on light', *The Observatory* **42**, 119–122.

Eigenbrod, A., Courbin, F., Vuissoz, C., Meylan, G., Saha, P. & Dye, S. (2005), 'Cosmograil: The cosmological monitoring of gravitational lenses-i. how to sample the light curves of gravitationally lensed quasars to measure accurate time delays', *Astronomy & Astrophysics* **436**(1), 25–35.

Eisenstein, D. J. (2005), 'Dark energy and cosmic sound', *New Astronomy Reviews* **49**(7-9), 360–365.

Ellis, G. F., Maartens, R. & MacCallum, M. A. (2012), *Relativistic cosmology*, Cambridge University Press.

Ellis, R. S. (2010), 'Gravitational lensing: a unique probe of dark matter and dark energy', *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* **368**(1914), 967–987.

Faure, C., Kneib, J.-P., Covone, G., Tasca, L., Leauthaud, A., Capak, P., Jahnke, K., Smolcic, V., de la Torre, S., Ellis, R., Finoguenov, A., Koekemoer, A., Fevre, O. L., Massey, R., Mellier, Y., Refregier, A., Rhodes, J., Scoville, N., Schinnerer, E., Taylor, J., Waerbeke, L. V. & Walcher, J. (2008), 'First catalog of strong lens candidates in the cosmos field', *The Astrophysical Journal Supplement Series* **176**(1), 19.
URL: *http://stacks.iop.org/0067-0049/176/i=1/a=19*

Golse, G., Kneib, J.-P. & Soucail, G. (2002), 'Constraining the cosmological parameters using strong lensing', *Astronomy & Astrophysics* **387**(3), 788–803.

Green, J., Schechter, P., Baltay, C., Bean, R., Bennett, D., Brown, R., Conselice, C., Donahue, M., Fan, X., Gaudi, B. et al. (2012), 'Wide-field infrared survey telescope (wfirst) final report', *arXiv preprint arXiv:1208.4012* .

Grillo, C., Lombardi, M. & Bertin, G. (2008), 'Cosmological parameters from strong gravitational lensing and stellar dynamics in elliptical galaxies', *Astronomy & Astrophysics* **477**(2), 397–406.

Heymans, C., Van Waerbeke, L., Miller, L., Erben, T., Hildebrandt, H., Hoekstra, H., Kitching, T. D., Mellier, Y., Simon, P., Bonnett, C. et al. (2012), 'Cfhtlens: the canada–france–hawaii telescope lensing survey', *Monthly Notices of the Royal Astronomical Society* **427**(1), 146–166.

Huang, K.-H., Ferguson, H. C., Ravindranath, S. & Su, J. (2013), 'The bivariate size-luminosity relations for lyman break galaxies at z 4-5', *The Astrophysical Journal* **765**(1), 68.

Ikarashi, S., Ivison, R., Caputi, K. I., Aretxaga, I., Dunlop, J. S., Hatsukade, B., Hughes, D. H., Iono, D., Izumi, T., Kawabe, R. et al. (2015), 'Compact starbursts in–6 submillimeter galaxies revealed by alma', *The Astrophysical Journal* **810**(2), 133.

Ivezic, Z., Tyson, J., Abel, B., Acosta, E., Allsman, R., AlSayyad, Y., Anderson, S., Andrew, J., Angel, R., Angeli, G. et al. (2008), 'Lsst: from science drivers to reference design and anticipated data products', *arXiv preprint arXiv:0805.2366* .

Jackson, N. (2008), 'Gravitational lenses and lens candidates identified from the cosmos field', *Monthly Notices of the Royal Astronomical Society* **389**(3), 1311–1318.
URL: *http://dx.doi.org/10.1111/j.1365-2966.2008.13629.x*

Jarvis, M. J., Bonfield, D. G., Bruce, V., Geach, J., McAlpine, K., McLure, R., González-Solares, E., Irwin, M., Lewis, J., Yoldas, A. K. et al. (2012), 'The vista deep extragalactic observations (video) survey', *Monthly Notices of the Royal Astronomical Society* **428**(2), 1281–1295.

Laureijs, R., Amiaux, J., Arduini, S., Augueres, J. ., Brinchmann, J., Cole, R., Cropper, M., Dabin, C., Duvet, L., Ealet, A. & et al. (2011), 'Euclid Definition Study Report', *ArXiv e-prints* .

Leaf, K. & Melia, F. (2018), 'Model selection with strong-lensing systems', *Monthly Notices of the Royal Astronomical Society* .

Li, N., Gladders, M. D., Heitmann, K., Rangel, E. M., Child, H. L., Florian, M. K., Bleem, L. E., Habib, S. & Finkel, H. J. (2018), 'The importance of secondary halos for strong lensing in massive galaxy clusters across redshift', *ArXiv e-prints* .

Li, Shun-Sheng., Mao, S., Zhao, Y. & Lu, Y. (2018), 'Gravitational lensing of gravitational waves: A statistical perspective', *Monthly Notices of the Royal Astronomical Society* **476**(2), 2220–2229.

Lintott, C. J., Schawinski, K., Slosar, A., Land, K., Bamford, S., Thomas, D., Raddick, M. J., Nichol, R. C., Szalay, A., Andreescu, D. et al. (2008), 'Galaxy zoo: morphologies derived from visual inspection of galaxies from the sloan digital sky survey', *Monthly Notices of the Royal Astronomical Society* **389**(3), 1179–1189.

Marshall, P. J., Verma, A., More, A., Davis, C. P., More, S., Kapadia, A., Parrish, M., Snyder, C., Wilcox, J., Baeten, E. et al. (2015), 'Space warps–i. crowdsourcing the discovery of gravitational lenses', *Monthly Notices of the Royal Astronomical Society* **455**(2), 1171–1190.

Massey, R., Kitching, T. & Richard, J. (2010), 'The dark matter of gravitational lensing', *Reports on Progress in Physics* **73**(8), 086901.

McCracken, H. J., Milvang-Jensen, B., Dunlop, J., Franx, M., Fynbo, J. P., Le Fèvre, O., Holt, J., Caputi, K. I., Goranova, Y., Buitrago, F. et al. (2013), 'Ultravista: A vista public survey of the distant universe', *The Messenger* **154**, 29–31.

Meneghetti, M. (2006), 'Introduction to gravitational lensing', *Lecture Notes (University of Heidelberg)* .

Misner, C. W., Thorne, K. S., Wheeler, J. A. & Gravitation, W. (1973), 'Freeman and company', *San Francisco* p. 891.

More, A., Verma, A., Marshall, P. J., More, S., Baeten, E., Wilcox, J., Macmillan, C., Cornen, C., Kapadia, A., Parrish, M. et al. (2015), 'Space warps–ii. new gravitational lens candidates from the cfhtls discovered through citizen science', *Monthly Notices of the Royal Astronomical Society* **455**(2), 1191–1210.

Mosleh, M., Williams, R. J., Franx, M., Gonzalez, V., Bouwens, R. J., Oesch, P., Labbe, I., Illingworth, G. D. & Trenti, M. (2012), 'The evolution of mass-size relation for lyman break galaxies from z= 1 to z= 7', *The Astrophysical Journal Letters* **756**(1), L12.

Narayan, R. & Bartelmann, M. (1996), 'Lectures on Gravitational Lensing', *ArXiv Astrophysics e-prints* .

Negrello, M., Amber, S., Amvrosiadis, A., Cai, Z.-Y., Lapi, A., Gonzalez-Nuevo, J., De Zotti, G., Furlanetto, C., Maddox, S., Allen, M. et al. (2017), 'The herschel-atlas: a sample of 500 μm-selected lensed galaxies over 600 deg2', *Monthly Notices of the Royal Astronomical Society* **465**(3), 3558–3580.

Negrello, M., Hopwood, R., De Zotti, G., Cooray, A., Verma, A., Bock, J., Frayer, D., Gurwell, M., Omont, A., Neri, R. et al. (2010*a*), 'The detection of a population of submillimeter-bright, strongly lensed galaxies', *science* **330**(6005), 800–804.

Negrello, M., Hopwood, R., De Zotti, G., Cooray, A., Verma, A., Bock, J., Frayer, D., Gurwell, M., Omont, A., Neri, R. et al. (2010*b*), 'The detection of a population of submillimeter-bright, strongly lensed galaxies', *science* **330**(6005), 800–804.

Oliver, S., Bock, J., Altieri, B., Amblard, A., Arumugam, V., Aussel, H., Babbedge, T., Beelen, A., Béthermin, M., Blain, A. et al. (2012), 'The herschel multi-tiered extragalactic survey: Hermes', *Monthly Notices of the Royal Astronomical Society* **424**(3), 1614–1635.

Price-Whelan, A. M., Sip'ocz, B. M., G"unther, H. M., Lim, P. L., Crawford, S. M., Conseil, S., Shupe, D. L., Craig, M. W., Dencheva, N., Ginsburg, A., VanderPlas, J. T., Bradley, L. D., P'erez-Su'arez, D., de Val-Borro, M., Paper Contributors, P., Aldcroft, T. L., Cruz, K. L., Robitaille, T. P., Tollerud, E. J., Coordination Committee, A., Ardelean, C., Babej, T., Bach, Y. P., Bachetti, M., Bakanov, A. V., Bamford, S. P., Barentsen, G., Barmby, P., Baumbach, A., Berry, K. L., Biscani, F., Boquien, M., Bostroem, K. A., Bouma, L. G., Brammer, G. B., Bray, E. M., Breytenbach, H., Buddelmeijer, H., Burke, D. J., Calderone, G., Cano Rodr'iguez, J. L., Cara, M., Cardoso, J. V. M., Cheedella, S., Copin, Y., Corrales, L., Crichton, D., D extquoterightAvella, D., Deil, C., Depagne, E., Dietrich, J. P., Donath, A., Droettboom, M., Earl, N., Erben, T., Fabbro, S., Ferreira, L. A., Finethy, T., Fox, R. T., Garrison, L. H., Gibbons, S. L. J., Goldstein, D. A., Gommers, R., Greco, J. P., Greenfield, P., Groener, A. M., Grollier, F., Hagen, A., Hirst, P., Homeier, D., Horton, A. J., Hosseinzadeh, G., Hu, L., Hunkeler, J. S., Ivezi'c, Z., Jain, A., Jenness, T., Kanarek, G., Kendrew, S., Kern, N. S., Kerzendorf, W. E., Khvalko, A., King, J., Kirkby, D., Kulkarni, A. M., Kumar, A., Lee, A., Lenz, D., Littlefair, S. P., Ma, Z., Macleod, D. M., Mastropietro, M., McCully, C., Montagnac, S., Morris, B. M., Mueller, M., Mumford, S. J., Muna, D., Murphy, N. A., Nelson, S., Nguyen, G. H., Ninan, J. P., N"othe, M., Ogaz, S., Oh, S., Parejko, J. K., Parley, N., Pascual, S., Patil, R., Patil, A. A., Plunkett, A. L., Prochaska, J. X., Rastogi, T., Reddy Janga, V., Sabater, J., Sakurikar, P., Seifert, M., Sherbert, L. E., Sherwood-Taylor, H., Shih, A. Y., Sick, J., Silbiger, M. T., Singanamalla, S., Singer, L. P., Sladen, P. H., Sooley, K. A., Sornarajah, S., Streicher, O., Teuben, P., Thomas, S. W., Tremblay, G. R., Turner, J. E. H., Terr'on, V., van Kerkwijk, M. H., de la Vega, A., Watkins, L. L., Weaver, B. A., Whitmore, J. B., Woillez, J., Zabalza, V. & Contributors, A. (2018), 'The Astropy Project: Building an Open-science Project and Status of the v2.0 Core Package', *aj* **156**, 123.

Refsdal, S. (1964), 'On the possibility of determining hubble's parameter and the masses of galaxies from the gravitational lens effect ', *Monthly Notices of the Royal Astronomical Society* **128**(4), 307–310.
URL: *http://dx.doi.org/10.1093/mnras/128.4.307*

Rhodes, J., Nichol, R. C., Aubourg, É., Bean, R., Boutigny, D., Bremer, M. N., Capak, P.,

Cardone, V., Carry, B., Conselice, C. J. et al. (2017), 'Scientific synergy between lsst and euclid', *The Astrophysical Journal Supplement Series* **233**(2), 21.

Riess, A. G., Macri, L., Casertano, S., Lampeitl, H., Ferguson, H. C., Filippenko, A. V., Jha, S. W., Li, W. & Chornock, R. (2011), 'A 3% solution: determination of the hubble constant with the hubble space telescope and wide field camera 3', *The Astrophysical Journal* **730**(2), 119.

Riess, A. G., Macri, L. M., Hoffmann, S. L., Scolnic, D., Casertano, S., Filippenko, A. V., Tucker, B. E., Reid, M. J., Jones, D. O., Silverman, J. M. et al. (2016), 'A 2.4% determination of the local value of the hubble constant', *The Astrophysical Journal* **826**(1), 56.

Schneider, P. (1992), Gravitational lensing statistics, *in* 'Gravitational Lenses', Springer, pp. 196–208.

Scott, P. (2003), 'Chi square: Testing for goodness of fit', *Lecture Notes (University of California Santa Cruz)* .
URL: *maxwell.ucsc.edu/ drip/133/ch4.pdf*

Scoville, N., Aussel, H., Brusa, M., Capak, P., Carollo, C., Elvis, M., Giavalisco, M., Guzzo, L., Hasinger, G., Impey, C. et al. (2007), 'The cosmic evolution survey (cosmos): overview', *The Astrophysical Journal Supplement Series* **172**(1), 1.

Serjeant, S. (2010), 'Observational cosmology', *Observational Cosmology by Stephen Serjeant. Cambridge University Press, 2010. ISBN: 9780521157155* .

Serjeant, S. (2012), 'Strong biases in infrared-selected gravitational lenses', *Monthly Notices of the Royal Astronomical Society* **424**(4), 2429–2441.

Smail, I., Ivison, R. & Blain, A. (1997), 'A deep submillimeter survey of lensing clusters: A new window on galaxy formation and evolution', *The Astrophysical Journal Letters* **490**(1), L5.

Soldner, J. (1804), 'On the deflection of a light ray from its rectilinear motion, by the attraction of a celestial body at which it nearly passes by', *Berliner Astronomisches Jahrbuch* pp. 161–172.

Suyu, S., Auger, M., Hilbert, S., Marshall, P., Tewes, M., Treu, T., Fassnacht, C., Koopmans, L., Sluse, D., Blandford, R. et al. (2013), 'Two accurate time-delay distances from strong lensing: Implications for cosmology', *The Astrophysical Journal* **766**(2), 70.

Suyu, S. H., Bonvin, V., Courbin, F., Fassnacht, C. D., Rusu, C. E., Sluse, D., Treu, T., Wong, K., Auger, M. W., Ding, X. et al. (2017), 'H0licow–i. h 0 lenses in cosmograil's wellspring: program overview', *Monthly Notices of the Royal Astronomical Society* **468**(3), 2590–2604.

Treu, T., Agnello, A., Baumer, M., Birrer, S., Buckley-Geer, E., Courbin, F., Kim, Y., Lin, H., Marshall, P., Nord, B. et al. (2018), 'The strong lensing insights into the dark energy survey (strides) 2016 follow-up campaign–i. overview and classification of candidates selected by two techniques', *Monthly Notices of the Royal Astronomical Society* **481**(1), 1041–1054.

Treu, T., Koopmans, L. V., Bolton, A. S., Burles, S. & Moustakas, L. A. (2006), 'The sloan lens acs survey. ii. stellar populations and internal structure of early-type lens galaxies', *The Astrophysical Journal* **640**(2), 662.

Walsh, D., Carswell, R. F. & Weymann, R. J. (1979), '0957+ 561 a, b: twin quasistellar objects or gravitational lens?', *Nature* **279**(5712), 381.

Wardlow, J. L., Cooray, A., De Bernardis, F., Amblard, A., Arumugam, V., Aussel, H., Baker, A., Béthermin, M., Blundell, R., Bock, J. et al. (2012), 'Hermes: candidate gravitationally lensed galaxies and lensing statistics at submillimeter wavelengths', *The Astrophysical Journal* **762**(1), 59.

Will, C. M. (2015), 'The 1919 measurement of the deflection of light', *Classical and Quantum Gravity* **32**(12), 124001.

Wyithe, J. S. B. & Loeb, A. (2011), 'Extrapolating the evolution of galaxy sizes to the epoch of reionization', *Monthly Notices of the Royal Astronomical Society: Letters* **413**(1), L38–L42.

Xu, Y.-Y. & Zhang, X. (2016), 'Comparison of dark energy models after planck 2015', *The European Physical Journal C* **76**(11), 588.

York, T., Jackson, N., Browne, I., Koopmans, L., McKean, J., Norbury, M., Biggs, A., Blandford, R., De Bruyn, A., Fassnacht, C. et al. (2005), 'Class b0631+ 519: last of the cosmic lens all-sky survey lenses', *Monthly Notices of the Royal Astronomical Society* **361**(1), 259–271.

Zavala, J. A., Montaña, A., Hughes, D. H., Yun, M. S., Ivison, R., Valiante, E., Wilner, D., Spilker, J., Aretxaga, I., Eales, S. et al. (2018), 'A dusty star-forming galaxy at z= 6 revealed by strong gravitational lensing', *Nature Astronomy* **2**(1), 56.

Zwicky, F. (1937), 'On the masses of nebulae and of clusters of nebulae', *The Astrophysical Journal* **86**, 217.